SZKOŁA GŁÓWNA HANDLOWA W WARSZAWIE
WARSAW SCHOOL OF ECONOMICS

Master's Studies

Field of studies: Advanced Analytics
Specialisation: Big Data

Author's Name and Surname: Oleksandr Romanchenko
Student No.: 83459

Deep Reinforcement Learning: training intelligent agent to play game "Flappy Bird" with evolution strategy algorithm

Master's thesis
Written in the Department/Institute[1]
……
Under scientific supervision of
dr. Jarosław Olejniczak

Warsaw 2019

Table of Contents

Introduction

The objective of this work is to demonstrate the possibilities of Reinforcement Learning technology on the example of mobile game "Flappy Bird". Why have I decided to choose the game instead of some business case from the real world? – Because games is the field where we have to find the optimal sequence of correct actions, constantly making decisions and it is the field where Reinforcement Learning thrives the most. In my experiment I'm going to use an algorithm called "Evolution Strategy" because it has proven to be less computationally demanding and more flexible in variety of Reinforcement Learning tasks and doesn't need to fit the MDP (Markov Decision Process) framework.

The structure is following:

- First, we are going to briefly discuss the history of machine learning
- Next, discuss different types of machine learning in order to acquire better understanding of what are the different purposes each of them is used for
- Then, we will take a closer look on the elements of reinforcement learning and practical cases of its implementation
- And finally show the practical implementation of Evolution Strategy algorithm on example of "Falppy Bird"

The idea of this thesis was inspired by artificial intelligence research company DeepMind which created a powerful AI (AlphaGo) based on Reinforcement Learning technology and trained it playing Go (ancient Chinese game). It showed amazing results and has beaten world champion Lee Sedol. After that Lee said that he has completely changed his view and understanding of the game. Also, the same technology was used by them to create Alpha Zero, the AI which has mastered playing chess.

Introduction to Machine Learning

We have seen Machine Learning as a buzzword for the past few years, the reason for this might be the high amount of data production by applications, the increase of computation power in the past few years and the development of better algorithms.

Machine Learning is used anywhere from automating mundane tasks to offering intelligent insights, industries in every sector try to benefit from it. We are already using devices that utilizes it. For example, a wearable fitness tracker like Fitbit, or an intelligent home assistant like Google Home. But there are much more examples of ML in use.

- Prediction — Machine learning can also be used in the prediction systems. Considering the loan example, to compute the probability of a fault, the system will need to classify the available data in groups.
- Image recognition — Machine learning can be used for face detection in an image as well. There is a separate category for each person in a database of several people.
- Speech Recognition — It is the translation of spoken words into the text. It is used in voice searches and more. Voice user interfaces include voice dialing, call routing, and appliance control. It can also be used a simple data entry and the preparation of structured documents.
- Medical diagnoses — ML is trained to recognize cancerous tissues.
- Financial industry and trading — companies use ML in fraud investigations and credit checks.

A Quick History of Machine Learning

It was in the 1940s when the first manually operated computer system, ENIAC (Electronic Numerical Integrator and Computer), was invented. At that time the word “computer” was being used as a name for a human with intensive numerical computation capabilities, so, ENIAC was called a numerical computing machine! Well, you may say it has nothing to do with learning?! WRONG, from the beginning the idea was to build a machine able to emulate human thinking and learning.

Figure 1. Electronic Numerical Integrator and Computer

source: www.computerhistory.org

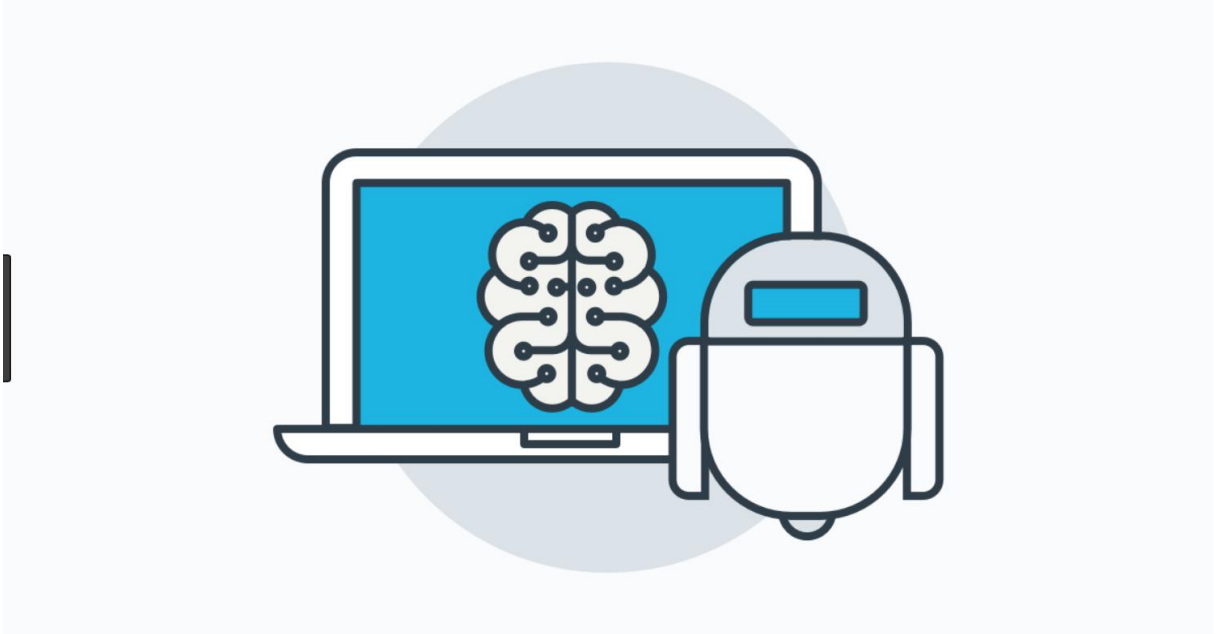
In the 1950s, we see the first computer game program claiming to be able to beat the checkers world champion. This program helped checkers players a lot in improving their skills! Around the same time, Frank Rosenblatt invented the Perceptron, which was a very, very simple classifier but when it was combined in large numbers, in a network, it became a powerful monster. Well, the monster is relative to the time and in that time, it was a real breakthrough. Then we see several years of stagnation of the neural network field due to its difficulties in solving certain problems.

Thanks to statistics, machine learning became very famous in the 1990s. The intersection of computer science and statistics gave birth to probabilistic approaches in AI. This shifted the field further toward data-driven approaches. Having large-scale data available, scientists started to build intelligent systems that were able to analyze and learn from large amounts of data. As a

highlight, IBM's Deep Blue system beat the world champion of chess, the grand-master Garry Kasparov. Kasparov accused IBM of cheating, but this is a piece of history now and Deep Blue is resting peacefully in a museum.

What is Machine Learning?

Figure 2



source: towardsdatascience.com

According to Arthur Samuel, Machine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed.

Machine learning (ML) is a category of an algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

Types of Machine Learning

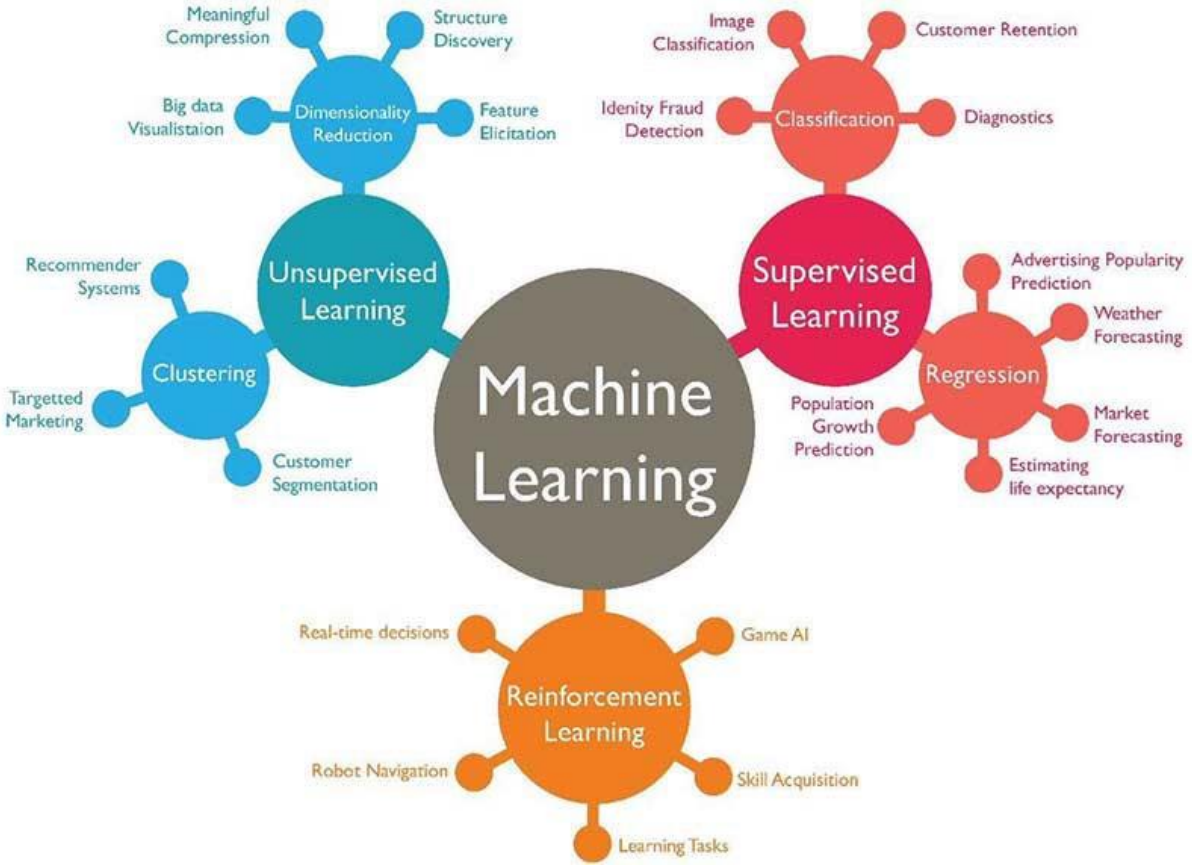
Machine learning can be classified into 3 types of algorithms.

1. Supervised Learning

2. Unsupervised Learning

3. Reinforcement Learning

Figure 3. 3 Types of Learning



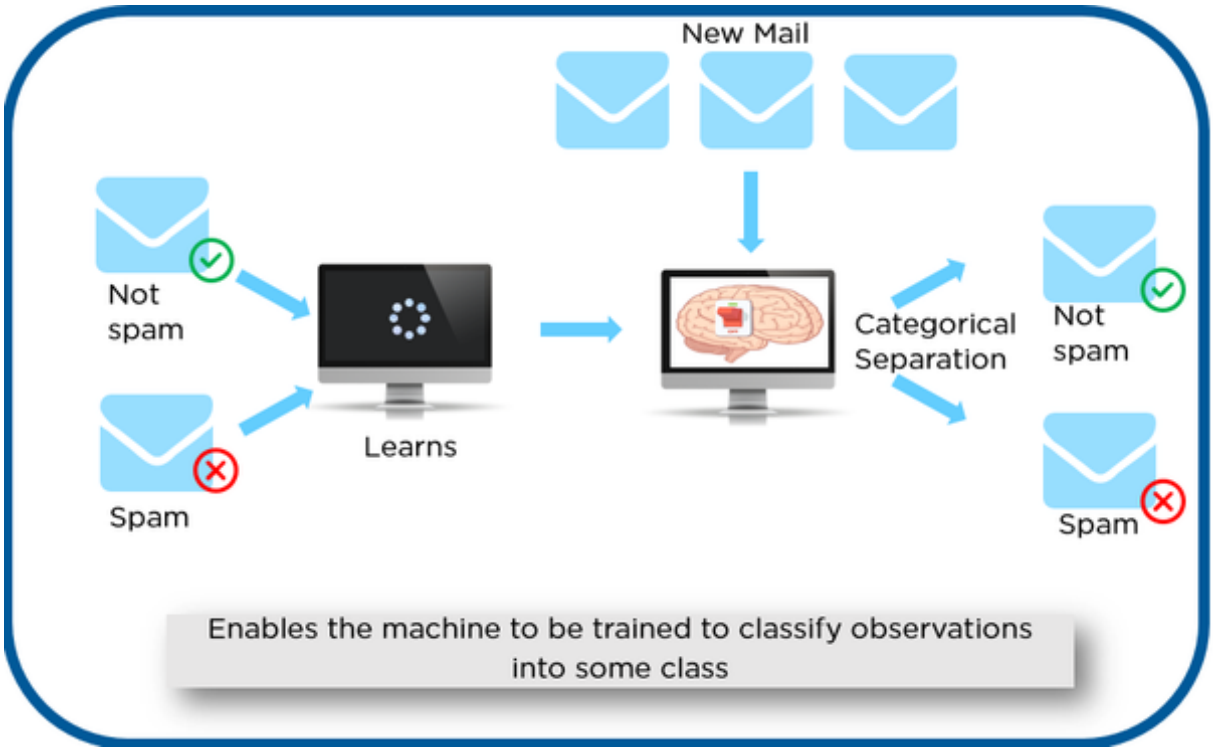
source: towardsdatascience.com

Overview of Supervised Learning Algorithm

In Supervised learning, an AI system is presented with data which is labeled, which means that each data tagged with the correct label.

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Figure 4. Example of Supervised Learning



source: towardsdatascince.com

As shown in the above example, we have initially taken some data and marked them as 'Spam' or 'Not Spam'. This labeled data is used by the training supervised model, this data is used to train the model.

Once it is trained, we can test our model by testing it with some test new mails and checking of the model is able to predict the right output.

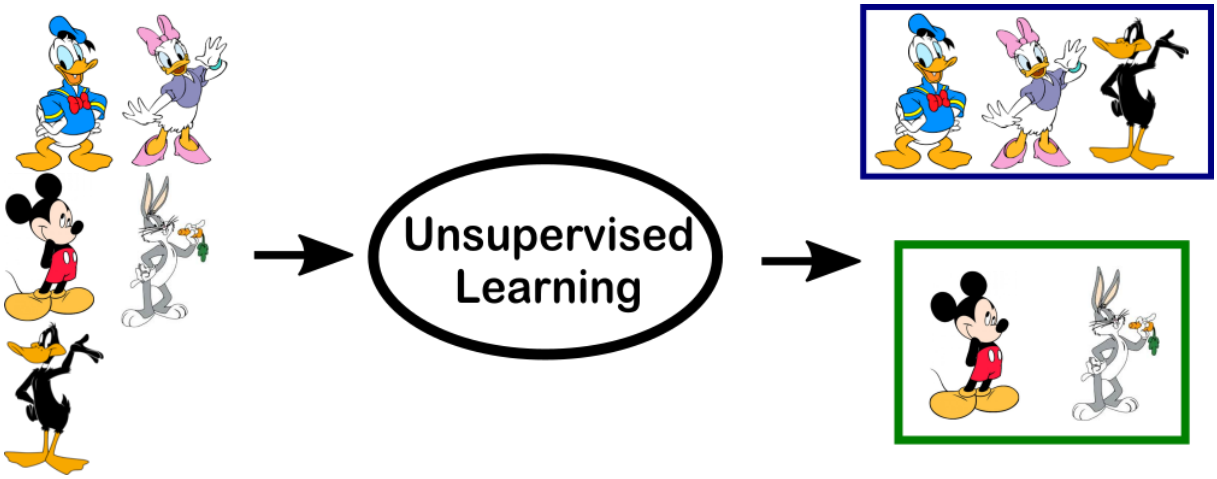
Types of Supervised learning

- **Classification**: A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease".
- **Regression**: A regression problem is when the output variable is a real value, such as "dollars" or "weight".

Overview of Unsupervised Learning Algorithm

In unsupervised learning, an AI system is presented with unlabeled, uncategorized data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.

Figure 5. Example of Unsupervised Learning



source: towardsdatascince.com

In the above example, we have given some characters to our model which are 'Ducks' and 'Not Ducks'. In our training data, we don't provide any label to the corresponding data. The unsupervised model is able to separate both the characters by looking at the type of data and models the underlying structure or distribution in the data in order to learn more about it.

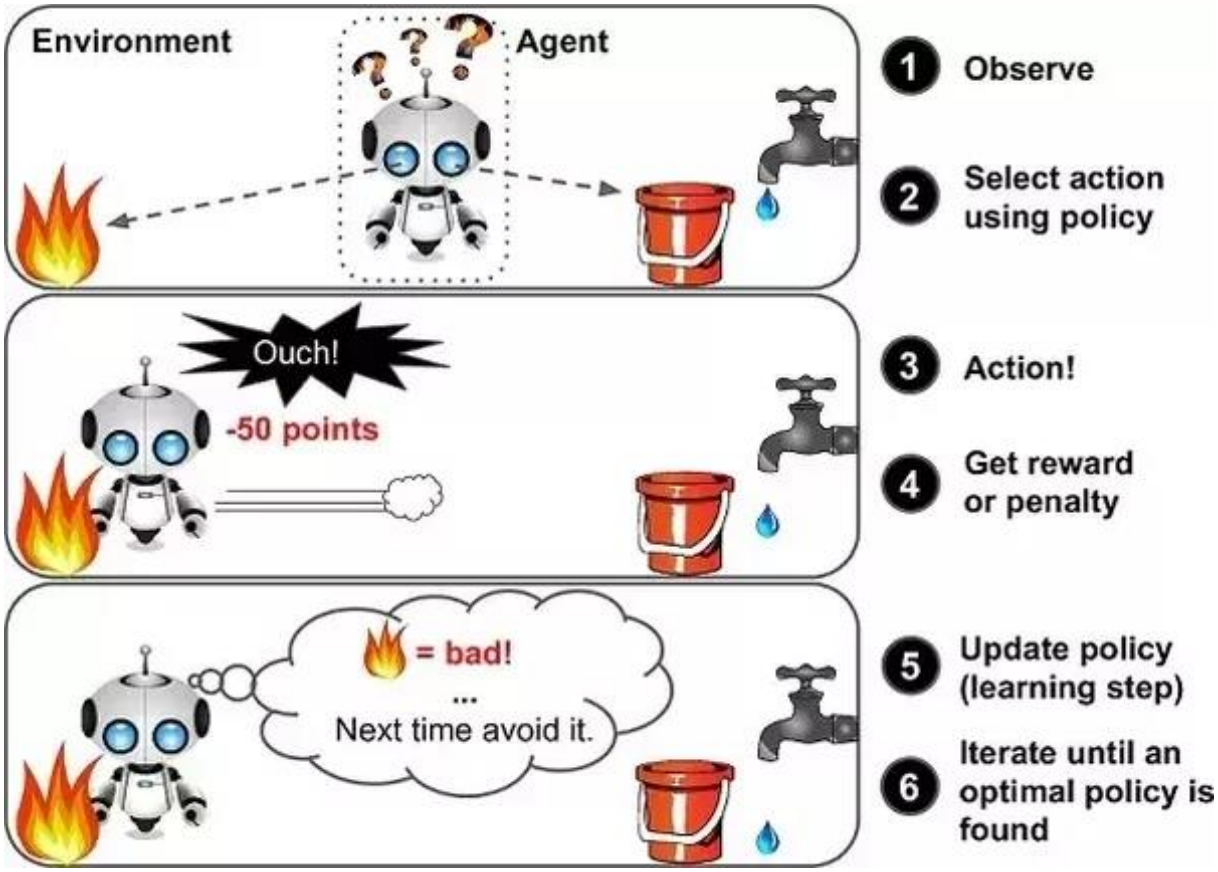
Types of Unsupervised learning

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Overview of Reinforcement Learning

A reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty. It is a type of dynamic programming that trains algorithms using a system of reward and punishment.

Figure 6. Example of Reinforcement Learning



source: towardsdatascince.com

In the above example, we can see that the agent is given 2 options i.e. a path with water or a path with fire. A reinforcement algorithm works on reward a system i.e. if the agent uses the fire path then the rewards are subtracted, and agent tries to learn that it should avoid the fire path. If it had chosen the water path or the safe path then some points would have been added to the reward points, the agent then would try to learn what path is safe and what path isn't. It is basically leveraging the rewards obtained, the agent improves its environment knowledge to select the next action.

What is deep learning?

Deep learning, on the other hand, is a subset of machine learning, which is inspired by the information processing patterns found in the human brain. The brain deciphers the information, labels it, and assigns it into different categories. When confronted with new information, the brain compares it with the existing information and arrives at the conclusion that spurs future action based on this analysis. Deep learning is based on numerous layers of algorithms (artificial neural networks) each providing a different interpretation of the data that's been fed to them.

How does deep learning work?

Before we tackle the question of "how it works," let's briefly define a few other necessary terms.

Supervised learning is using labeled data sets that have inputs and expected outputs. Unsupervised learning is using data sets with no specified structure.

In the case of supervised learning, a user is expected to train the AI to make the right decision: the user gives the machine the input and the expected output, if the output of AI is wrong, it will readjust its calculations; the iterative process goes on until the AI makes no more mistakes. Among the popular supervised algorithms are linear regression, logistic regression, decision trees, support vector machines, and non-parametric models such as k-Nearest Neighbors. In the case of unsupervised learning, the user lets the AI make logical classifications from the data. Here, algorithms such as hierarchical clustering, k-Means, Gaussian mixture models attempt to learn meaningful structures in the data.

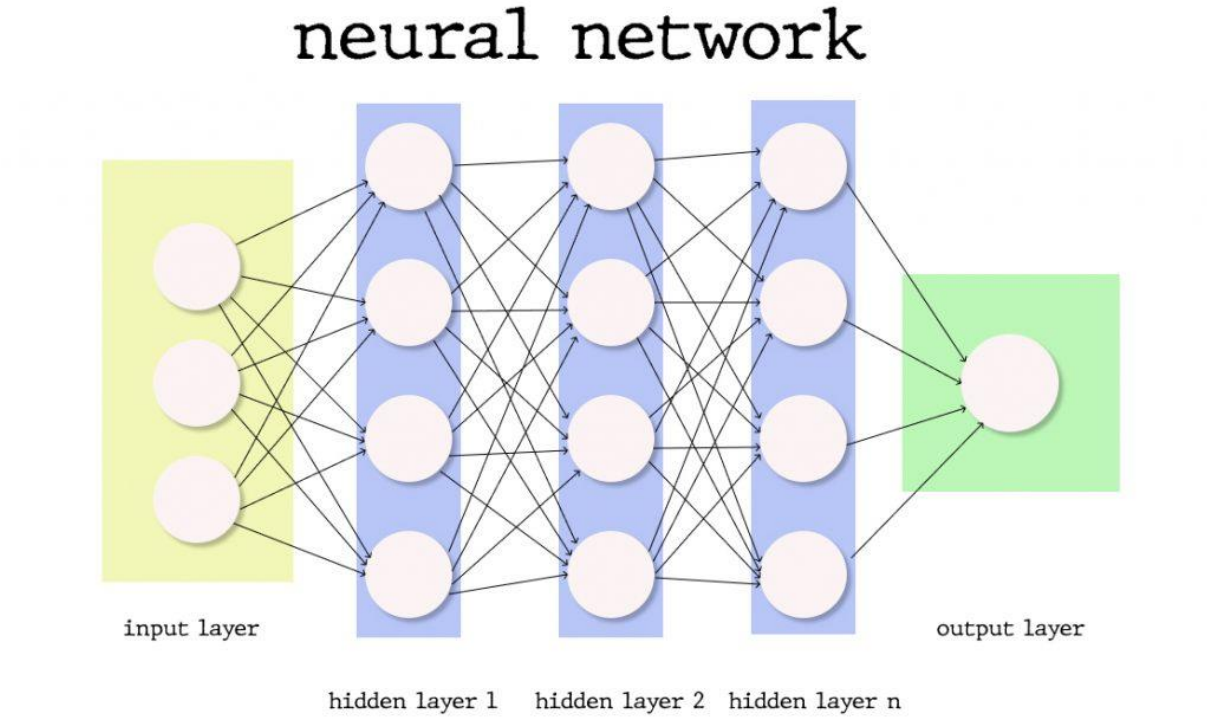
Deep Learning operates without strict rules as the ML algorithms should extract the trends and patterns from the vast sets of unstructured data after accomplishing the process of either supervised or unsupervised learning. To proceed further, we'll need to define neural networks.

Deep Learning vs Neural Network

The Deep Learning underlying algorithm is neural networks — the more layers, the deeper the network. A layer consists of computational nodes, "neurons," every one of which connects to all of the neurons in the underlying layer. There are three types of layers:

- The input layer of nodes, which receive information and transfers it to the underlying nodes
- Hidden node layers are the ones which take all calculations
- Output node layer is a place for computational results

Figure 7. Neural Network



source: towardsdatascince.com

By adding more hidden layers into the network, the researchers enable more in-depth calculations; however, the more layers — the more computational power is needed to deploy such a network.

Each connection has its weight and importance, the initial values of which are assigned randomly or according to their perceived importance for the ML model training dataset creator.

The activation function for every neuron evaluates the way the signal should be taken, and if the data analyzed differs from the expected, the weight values are configured anew, and the iteration begins. The difference between the yielded results and the expected is called the loss function, which we need to be as close to zero as possible. Gradient Descent is a function that describes how changing connection importance affects output accuracy. After each iteration, we adjust the weights of the nodes in small increments and find out the direction to reach the set minimum. After several of said iterations, the trained Deep Learning model is expected to produce relatively accurate results and can be deployed to production, however, some tweaking and adjustments can be necessary if the weight of the factors change over time.

Deep learning Learning Overview: summary of how DL works

Deep Learning is one of the ways of implementing Machine Learning through artificial neural networks, algorithms that mimic the structure of the human brain. Basically, DL algorithms use multiple layers to progressively extract higher-level features from the raw input. In DL, each level learns to transform its input data into more abstract representation, more importantly, a deep learning process can learn which features to optimally place in which level on its own, without human interaction. DL is both applicable for supervised and unsupervised learning tasks, where for supervised tasks DL methods eliminate feature engineering and derive layered structures that remove redundancy in representation; DL structures that can be used in an unsupervised manner are deep belief networks and neural history compressors.

Deep Learning Applications

Now, let’s look at some of the top applications of deep learning, which will help you better understand DL and how it works, besides some of those offer fantastic tutorials and source code detailing how to implement those algorithms.

The most well-known application of deep learning is a recommendation engine that’s supposed to enhance the user experience and provide a better service to its users. There are two types of recommendation engines: content-based and collaborative filtering. Until you have a sizable user-base, it’s best recommended to start with the content-based engine first.
Natural Language Processing and Recurrent Neural Networks are used in the text to extract higher level information, also known as text sentiment analysis.

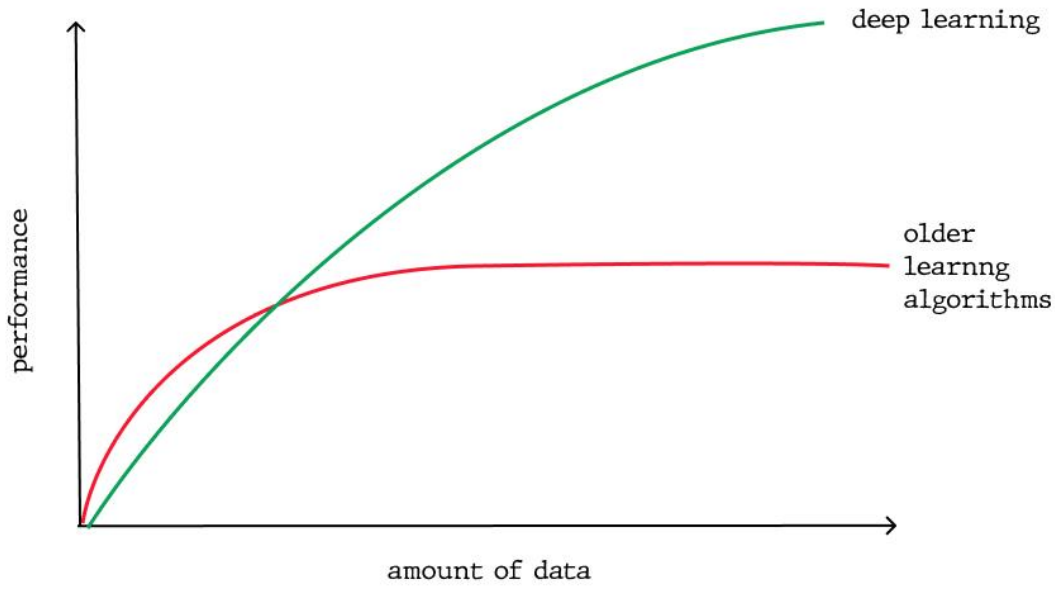
Another popular application is chatbots that can be trained with samples of dialogs and recurrent neural networks

Another popular application of DL models is image retrieval and classification using recognition models to sort images into different categories or using auto-encoders to retrieve images based on visual similarity.

Machine Learning vs Deep Learning: comparison

One of the most important differences is in the scalability of deep learning versus older machine learning algorithms: when data is small, deep learning doesn't perform well, but as the amount of data increases, deep learning skyrockets in understanding and performing on that data; conversely, traditional algorithms don't depend on the amount of data as much.

Figure 8. Scaling with Amount of Data



source: towardsdatascince.com

Another important distinction which ensues directly from the first difference is the deep learning hardware dependency: Dl algorithms depend on high-end machines and GPUs, because they do a large amount of matrix multiplication operations, whereas older machine learning algorithms can work on low-end machines perfectly well.

In machine learning, most of the applied features need to be identified by a machine learning expert, who then hand-copies them as per domain and data type. The input values (or features) can be anything from pixel values, shapes, textures, etc. The performance of the older ML algorithm will thus depend largely on how well and accurately the features were inputted, identified, extracted. Deep Learning learns high-level features from data, this is a major shift from traditional ML since it reduces the task of developing new feature extractor for every problem, in turn, DL will learn low features in early layers of the neural network and then high-level as it goes deeper into the said network.

Again, because of the large amount of data that needs to be learned from, deep learning algorithms take quite a lot of time to train, sometimes as long as several weeks, comparatively, machine learning takes much less time to train to range from a second to a few hours. However, during the testing time, deep learning takes less time to run than an average machine learning algorithm.

Also, interpretability is a factor for comparison. With deep learning algorithms, sometimes it's impossible to interpret the results, that's exactly why some industries have had slow adoptions of DL. Nevertheless, DL models can still achieve high accuracy but at the cost of higher abstraction. To elaborate on this a little further, let's get back to the weights in a neural network (NN), which essentially indicates a measure of how strong each connection is between each neuron. So by looking at the first layer, you can tell how strong is the connection between the inputs and the first layer's neurons. But at the second level, you'll lose the relationship, because the one-to-many relationship has turned into many-to-many relationships, exactly because of the high complexity of the NN nature: a neuron in one layer can be related to some other neurons which are far away from the first layer, deep into the network. Again, weights tell the story about the input, but that information is compressed after the application of the activation functions making it near impossible to decode. On the other hand, machine learning algorithms like decision trees give explicit rules as to why it chose what it chose and thus, they are easier to interpret.

Theory behind Reinforcement Learning

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behaviour. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence. Rather than directly theorizing about how people or animals learn, we explore idealized learning situations and evaluate the effectiveness of various learning methods. That is, adopting the perspective of an artificial intelligence researcher or engineer. We explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating the designs through mathematical analysis or computational experiments. The approach we explore, called reinforcement learning, is much more focused on goal-directed learning from interaction than are other approaches to machine learning.

Reinforcement learning is like many topics with names ending in -ing, such as machine learning, planning, and mountaineering, in that it is simultaneously a problem, a class of solution methods that work well on the class of problems, and the field that studies these problems and their solution methods. Reinforcement learning problems involve learning what to do - how to map situations to actions so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics being closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions, including reward signals, play out over extended time periods are the three most important distinguishing features of reinforcement learning problems. A full specification of reinforcement learning problems in

terms of optimal control described with Markov decision processes which idea is simply to capture the most important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal. Clearly, such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment. The formulation is intended to include just these three aspects: sensation, action, and goal in their simplest possible forms without trivializing any of them. Any method that is well suited to solving this kind of problem we consider to be a reinforcement learning method. Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in field of machine learning. Supervised learning is learning from a training set of labelled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification the label of the correct action the system should take to that situation, which is often to identify a category to which the situation belongs. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behaviour that are both correct and representative of all the situations in which the agent has to act. In uncharted territory where one would expect learning to be most beneficial an agent must be able to learn from its own experience. Reinforcement learning is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabelled data.

The terms supervised learning and unsupervised learning appear to exhaustively classify machine learning paradigms, but they do not. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behaviour, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning agent's problem of maximizing a reward signal. We therefore consider reinforcement learning to be a third machine learning paradigm, alongside of supervised learning, unsupervised learning, and perhaps other paradigms as well. One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such

actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate its expected reward. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. This is in contrast with many approaches that consider subproblems without addressing how they might _t into a larger picture. For example, we have mentioned that much of machine learning research is concerned with supervised learning without explicitly specifying how such an ability would
finally be useful. Other researchers have developed theories of planning with general goals, but without considering planning's role in real-time decision-making, or the question of where the predictive models necessary for planning would come from. Although these approaches have yielded many useful results, their focus on isolated subproblems is a significant limitation. Reinforcement learning takes the opposite tack, starting with a complete, interactive, goal-seeking agent. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to in influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate despite significant uncertainty about the environment it faces. When reinforcement learning involves planning, it has to address the interplay between planning and real-time action selection, as well as the question of how environment models are acquired and improved. When reinforcement learning involves supervised learning, it does so for specific reasons that determine which capabilities are critical and which are not. For learning
research to make progress, important subproblems have to be isolated and studied, but they should be subproblems that play clear roles in complete, interactive, goal-seeking agents, even if all the details of the complete agent cannot yet be filled in. One of the most exciting aspects of modern reinforcement learning is its substantive and fruitful interactions with other engineering and scientific disciplines. Reinforcement learning is part of a decades-long trend within artificial intelligence and machine learning toward greater integration with statistics,
optimization, and other mathematical subjects. For example, the ability of some reinforcement learning methods to learn with parameterized approximators addresses the classical \curse of

dimensionality" in operations research and control theory. More distinctively, reinforcement learning has also interacted strongly with psychology and neuroscience, with substantial benefits going both ways. Of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems. And reinforcement learning has also given back, both through a psychological model of animal learning that better matches some of the empirical data, and through an influential model of parts of the brain's reward system. Finally, reinforcement learning is also part of a larger trend in artificial intelligence back toward simple general principles. Since the late 1960's, many artificial intelligence researchers presumed that there are no general principles to be discovered, that intelligence is instead due to the possession of vast numbers of special purpose tricks, procedures, and heuristics. It was sometimes said that if we could just get enough relevant facts into a machine, say one million, or one billion, then it would become intelligent. Methods based on general principles, such as search or learning, were characterized as “weak methods”, whereas those based on specific knowledge were called “strong methods." This view is still common today, but much less dominant. Modern AI now includes much research looking for general principles of learning, search, and decision-making, as well as trying to incorporate vast amounts of domain knowledge. It is not clear how far back the pendulum will swing, but reinforcement learning research is certainly part of the swing back toward simpler and fewer general principles of artificial intelligence.

Examples of application of reinforcement learning

A good way to understand reinforcement learning is to consider some of the examples and possible applications that have guided its development.

- A master chess player makes a move. The choice is informed both by planning/anticipating possible replies and counterreplies and by immediate, intuitive judgments of the desirability of particular positions and moves.
- An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. The controller optimizes the yield/cost/quality trade-off on the basis of specified marginal costs without sticking strictly to the set points originally suggested by engineers.
- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.

- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.
- Phil prepares his breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behaviour and interlocking goal-subgoal relationships: walking to the cupboard, opening it, selecting a cereal box, then reaching for, grasping, and retrieving the box. Other complex, tuned, interactive sequences of behaviour are required to obtain a bowl, spoon, and milk jug. Each step involves a series of eye movements to obtain information and to guide reaching and locomotion. Rapid judgments are continually made about how to carry the objects or whether it is better to ferry some of them to the dining table before obtaining others. Each step is guided by goals, such as grasping a spoon or getting to the refrigerator, and is in service of other goals, such as having the spoon to eat with once the cereal is prepared and ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about the state of his body that determines his nutritional needs, level of hunger, and food preferences. These examples share features that are so basic that they are easy to overlook. All involve interaction between an active decision-making agent and its environment, within which the agent seeks to achieve a goal despite uncertainty about its environment. The agent's actions are permitted to affect the future state of the environment (e.g., the next chess position, the level of reservoirs of the refinery, the robot's next location and the future charge level of its battery), thereby affecting the options and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning.

At the same time, in all these examples the effects of actions cannot be fully predicted; thus the agent must monitor its environment frequently and react appropriately. For example, Phil must watch the milk he pours into his cereal bowl to keep it from overflowing. All these examples involve goals that are explicit in the sense that the agent can judge progress toward its goal based on what it can sense directly. The chess player knows whether or not he wins, the refinery controller knows how much petroleum is being produced, the mobile robot knows when its batteries run down, and Phil knows whether or not he is enjoying his breakfast. Neither the agent nor its environment may coincide with what we normally think of as an agent and its environment. An agent is not necessarily an entire robot or organism, and its environment is not necessarily only what is outside of a robot or organism. The example robot's battery is part of the environment of its controlling agent, and Phil's degree of hunger and food preferences

are features of the environment of his internal decision-making agent. The state of an agent's environment often includes information about the state of the machine or organism in which the agent resides, and this can include memories and even aspirations.
In all of these examples the agent can use its experience to improve its performance over time. The chess player refines the intuition he uses to evaluate positions, thereby improving his play; the gazelle calf improves the efficiency with which it can run; Phil learns to streamline making his breakfast. The knowledge the agent brings to the task at the start either from previous experience with related tasks or built into it by design or evolution - influences what is useful or easy to learn, but interaction with the environment is essential for adjusting behaviour to exploit specific features of the task.

Elements of Reinforcement Learning

Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment. A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus-response rules or associations (provided that stimuli include those that can come from within the animal). In some cases, the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behaviour. In general, policies may be stochastic. A reward signal defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number, a reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward sent to the agent at any time depends on the agent's current action and the current state of the agent's environment. The agent cannot alter the process that does this. The only way the agent can influence the reward signal is through its actions, which can have a direct effect on reward, or an indirect effect through changing the environment's state. In our example above of Phil eating breakfast, the reinforcement learning agent directing his behaviour might receive different reward signals when he eats his breakfast

depending on how hungry he is, what mood he is in, and other features of his of his body, which is part of his internal reinforcement learning agent's environment. The reward signal is the primary basis for altering the policy. If an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken. Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the

immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and

farsighted judgment of how pleased or displeased we are that our environment is in a particular state. Expressed this way, it is clear that value functions formalize a basic and familiar idea. Rewards are in a sense primary, whereas values, as predictions of rewards,

are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. In decision-making and

planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms we consider is a

method for efficiently estimating values. The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades.

The fourth and final element of some reinforcement learning systems is a model of the environment. This is something that mimics the behaviour of the environment, or more generally, that allows inferences to be made about how the environment will behave. For

example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and-error learners viewed as almost the opposite of planning.

Limitations and Scope

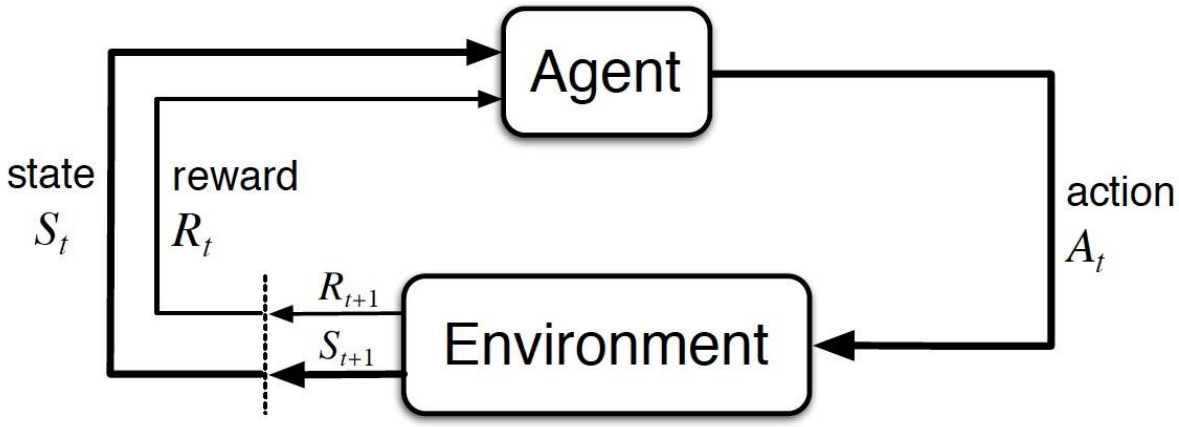
Most of the reinforcement learning methods are structured around estimating value functions, but it is not strictly necessary to do this to solve reinforcement learning problems. For example, methods such as genetic algorithms, genetic programming, simulated annealing, and other optimization methods have been used to approach reinforcement learning problems without ever appealing to value functions. These methods evaluate the lifetime behaviour of many non-learning agents, each using a different policy for interacting with its environment, and select those that are able to obtain the most reward. We call these evolutionary methods because their operation is analogous to the way biological evolution produces organisms with skilled behavior even when they do not learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are common or easy to find or if a lot of time is available for the search then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment. Evolutionary methods ignore much of the useful structure of the reinforcement learning problem: they do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases this information can be misleading (e.g., when states are misperceived), but more often it should enable more efficient search. However, there are some methods that, like evolutionary methods, do not appeal to value functions. These methods search in spaces of policies defined by a collection of numerical parameters. They estimate the directions the parameters should be adjusted in order to most rapidly improve a policy's performance. Unlike evolutionary methods, however, they produce these estimates while the agent is interacting with its environment and so can take advantage of the details of individual behavioral interactions. Methods like this, called policy gradient methods, have proven useful in many problems, and some of the simplest reinforcement learning methods fall into this category. In fact, some of these methods take

advantage of value function estimates to improve their gradient estimates. Overall, the distinction between policy gradient methods and other methods we include as reinforcement learning methods is not sharply defined. Reinforcement learning's connection to optimization methods deserves some additional comment because it is a source of a common misunderstanding. When we say that a reinforcement learning agent's goal is to maximize a numerical reward signal, we of course are not insisting that the agent has to actually achieve the goal of maximum reward. Trying to maximize a quantity does not mean that that quantity is ever maximized. The point is that a reinforcement learning agent is always trying to increase the amount of reward it receives. Many factors can prevent it from achieving the maximum, even if one exists. In other words, optimization is not the same as optimality.

The Agent-Environment Interface

The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also

Figure 9. The agent-environment interaction in reinforcement learning



source: Reinforcement Learning: Theory and Practice (Csaba Szepesvari)

gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem. More specifically, the agent and environment interact at each of a sequence

of discrete time steps, t = 0, 1, 2, 3... At each time step t, the agent receives some representation of the environment's state, $S_t \in S$, where S is the set of possible states, and on that basis selects an action, $A_t \in A(S_t)$, where $A(S_t)$, is the set of actions available in state S_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(a|s)$, is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning methods specify how the agent changes its policy as

a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run. This framework is abstract and flexible and can be applied to many different problems in many different ways. For example, the time steps need not refer to fixed intervals of real time; they can refer to arbitrary successive stages of decision-making and acting. The actions can be low-level controls, such as the voltages applied to the motors of a robot arm, or high-level decisions, such as whether or not to have lunch or to go to graduate school. Similarly, the states can take a wide variety of forms. They can be completely determined by low-level sensations, such as direct sensor readings, or they can be more high-level and abstract, such as symbolic descriptions of objects in a room.

Some of what makes up a state could be based on memory of past sensations or even be entirely mental or subjective. For example, an agent could be in the state of not being sure where an object is, or of having just been surprised in some clearly defined sense. Similarly, some actions might be totally mental or computational. For example, some actions might control what an agent chooses to think about, or where it focuses its attention. In general, actions can be any decisions we want to learn how to make, and the states can be

anything we can know that might be useful in making them. In particular, the boundary between agent and environment is not often the same as the physical boundary of a robot's or animal's body. Usually, the boundary is drawn closer to the agent than that. For example, the motors and mechanical linkages of a robot and its sensing hardware should usually be

considered parts of the environment rather than parts of the agent. Similarly, if we apply the framework to a person or animal, the muscles, skeleton, and sensory organs should be considered part of the environment. Rewards, too, presumably are computed inside the physical bodies of natural and artificial learning systems, but are considered external to the agent. The general rule we follow is that anything that cannot be changed arbitrarily

by the agent is considered to be outside of it and thus part of its environment. We do not assume that everything in the environment is unknown to the agent. For example, the agent often knows

quite a bit about how its rewards are computed as a function of its actions and the states in which they are taken. But we always consider the reward computation to be external to the agent because it defines the task facing the agent and thus must be beyond its ability to change arbitrarily. In fact, in some cases the agent may know everything about how its environment works and still face a difficult reinforcement learning task, just as we may know exactly how a puzzle like Rubik's cube works, but still be unable to solve it. The agent-environment boundary represents the limit of the agent's absolute control, not of its knowledge.

The agent-environment boundary can be located at different places for different purposes. In a complicated robot, many different agents may be operating at once, each with its own boundary. For example, one agent may make high-level decisions which form part of the states faced by a lower-level agent that implements the high-level decisions. In practice, the agent-environment boundary is determined once one has selected particular states, actions, and rewards, and thus has identified a specific decision-making task of interest. The reinforcement learning framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behaviour can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable. Of course, the particular states and actions vary greatly from task to task, and how they are represented can strongly affect performance. In reinforcement learning, as in other kinds of learning, such representational choices are at present more art than science.

Goals and Rewards

In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent. At each time step, the reward is a simple number, $R_t \in \mathbb{R}$. Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run. We can clearly state this informal idea as the reward hypothesis:

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. Although formulating goals in terms of reward signals might at first appear limiting, in practice it has proved to be flexible and widely applicable. The best way to see this is to consider examples of how it has been, or could be, used. For example, to make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion. In making a robot learn how to escape from a maze, the reward is often -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible. To make a robot learn to find and collect empty soda cans for recycling, one might give it a reward of zero most of the time, and then a reward of +1 for each can collected. One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it. For an agent to learn to play checkers or chess, the natural rewards are +1 for winning, -1 for losing, and 0 for drawing and for all nonterminal positions. You can see what is happening in all of these examples. The agent always learns to maximize its reward. If we want it to do something for us, we must

provide rewards to it in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want accomplished. In particular, the reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do. For example, a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such taking its opponent's pieces or gaining control of the center of the board. If achieving these sorts of subgoals were rewarded, then the agent might find a way to achieve them without achieving the real goal. For example,

it might find a way to take the opponent's pieces even at the cost of losing the game. The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.

Newcomers to reinforcement learning are sometimes surprised that the rewards which define of the goal of learning are computed in the environment rather than in the agent. Certainly most ultimate goals for animals are recognized by computations occurring inside their bodies, for example, by sensors for recognizing food, hunger, pain, and pleasure. Nevertheless, as we discussed in the previous section, one can redraw the agent-environment interface in such a way that these parts of the body are considered to be outside of the agent (and thus part of the agent's environment). For example, if the goal concerns a robot's internal energy reservoirs, then these are considered to be part of the environment; if the goal concerns the positions of the robot's limbs, then these too are considered to be part of the environment that is, the agent's boundary is drawn at the interface between the limbs and their control systems. These things are considered internal to the robot but external to the learning agent. For our purposes, it is convenient to place the boundary of the learning agent not at the limit of its physical body, but at the limit of its control. The reason we do this is that the agent's ultimate goal should be something over which it has imperfect control: it should not be able, for example, to simply decree that the reward has been received in the same way that it might arbitrarily change its actions. Therefore, we place the reward source outside of the agent. This does not preclude the agent from defining for itself a kind of internal reward, or a sequence of internal rewards. Indeed, this is exactly what many reinforcement learning methods do.

Returns (G_t) – cumulative reward the agent receives in the long run.

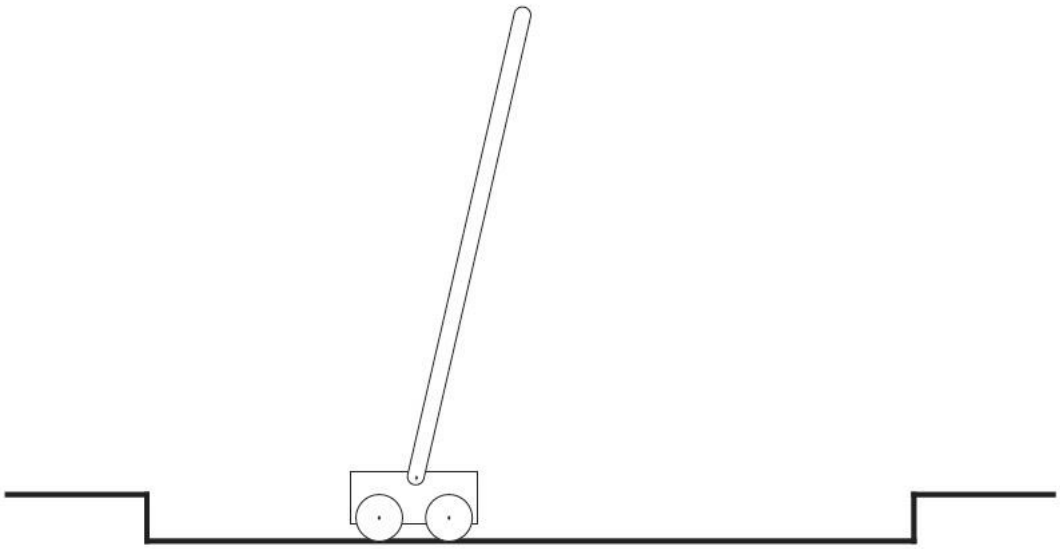
So far we have discussed the objective of learning informally. We have said that the agent's goal is to maximize the cumulative reward it receives in the long run. How might this be defined formally? If the sequence of rewards received after time step t is denoted $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$, then what precise aspect of this sequence do we wish to maximize? In general, we seek to maximize the expected return, where the return G_t is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \qquad [1.0]$$

where T is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into subsequences, which we call episodes, such as plays of a game, trips through a maze, or any sort of repeated interactions. Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Tasks with episodes of this kind are called episodic tasks. In episodic tasks we sometimes need to distinguish the set of all nonterminal states, denoted S, from the set of all states plus the terminal state, denoted S^+.

On the other hand, in many cases the agent-environment interaction does not break naturally into identifiable episodes but goes on continually without limit. For example, this would be the natural way to formulate a continual process-control task, or an application to a robot with a long-life span. We call these continuing tasks. The return formulation is problematic for continuing tasks because the final time step would be T = 1, and the return, which is what we are trying to maximize, could itself easily be infinite. (For example, suppose the agent receives a reward of +1 at each time step.)

Figure 10. The pole-balancing task



source: Reinforcement Learning: Theory and Practice (Csaba Szepesvari)

The additional concept that we need is that of discounting. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses At to maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad [1.1]$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate. The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. If $\gamma < 1$, the infinite sum has an infinite value as long as the reward sequence $\{R_k\}$ is bounded. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose A_t so as to maximize only R_{t+1}. If each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize by separately maximizing each immediate reward. But in general, acting to maximize immediate reward can reduce access to future rewards so that the return may actually be reduced. As γ approaches 1, the objective takes future rewards into account more strongly: the agent becomes more farsighted.

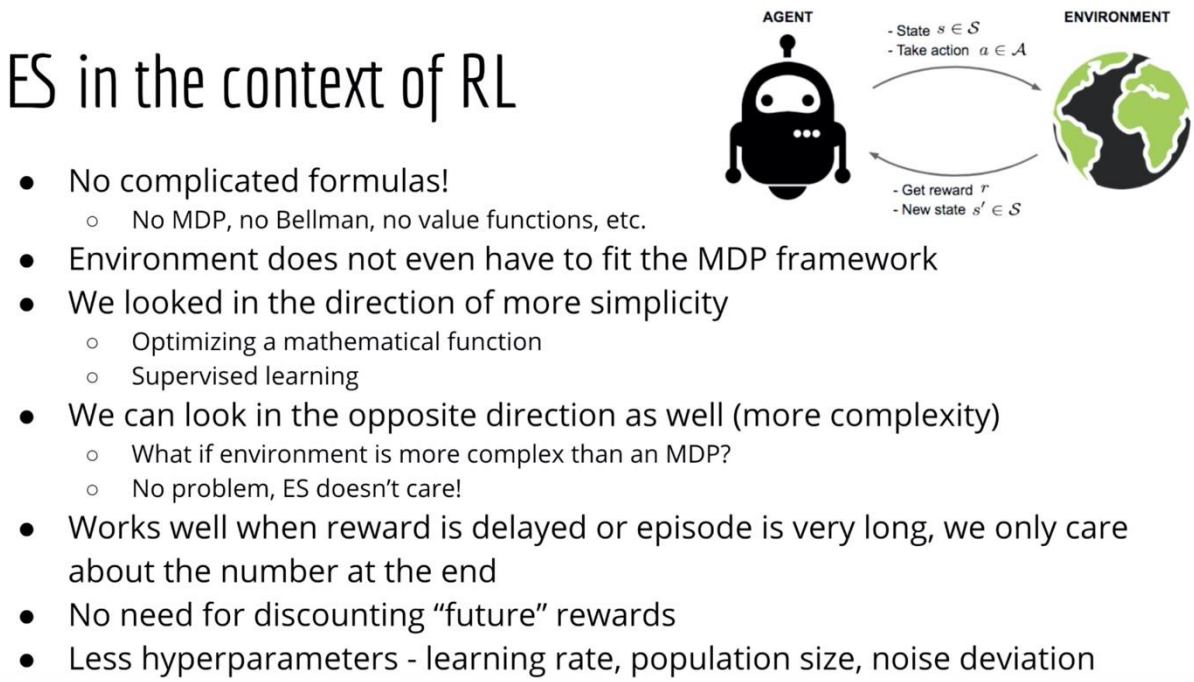
Implementation of agent playing "Flappy Bird" game

Evolution strategy:

For our experiment we are going to use evolution strategy algorithm because it has proven to be less computationally demanding and more flexible in variety of Reinforcement Learning tasks and doesn't need to fit the MDP framework.

Here are some benefits which brings to us evolution strategy:

Figure 11. Evolution strategy in the context of Reinforcement Learning

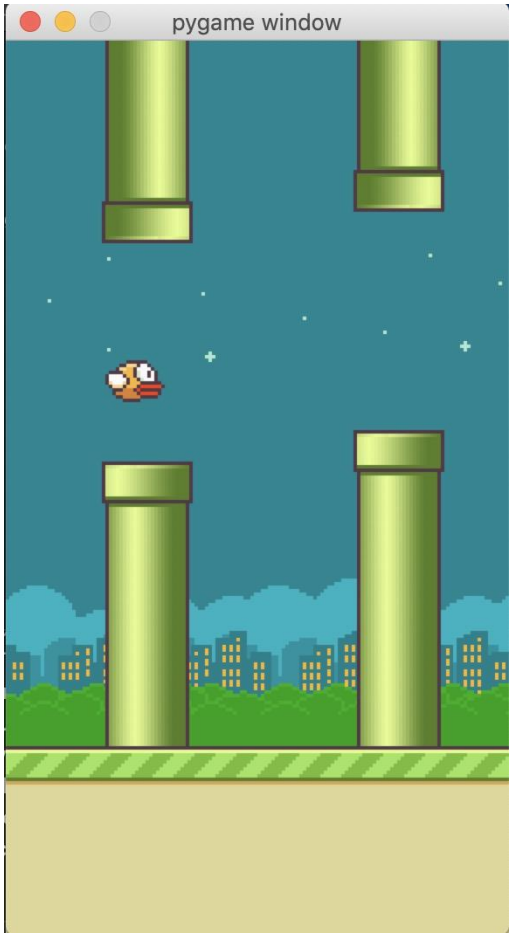


source: udemy.com

Our environment is a mobile game Flappy Bird where the player (or the agent in our case) has to maintain the position of the bird as long as it can. Every time the bird falls down – game starts from the beginning. The task may seem difficult for humans but not for the agent.

This is how the game looks like:

Figure 12. Illustration of the game



source: own elaboration

All the agent needs to do is to continuously fly between these pipes. Each time passing a “gate” formed by a pair of pipes player gets +1 score.

Evolution Strategy theory

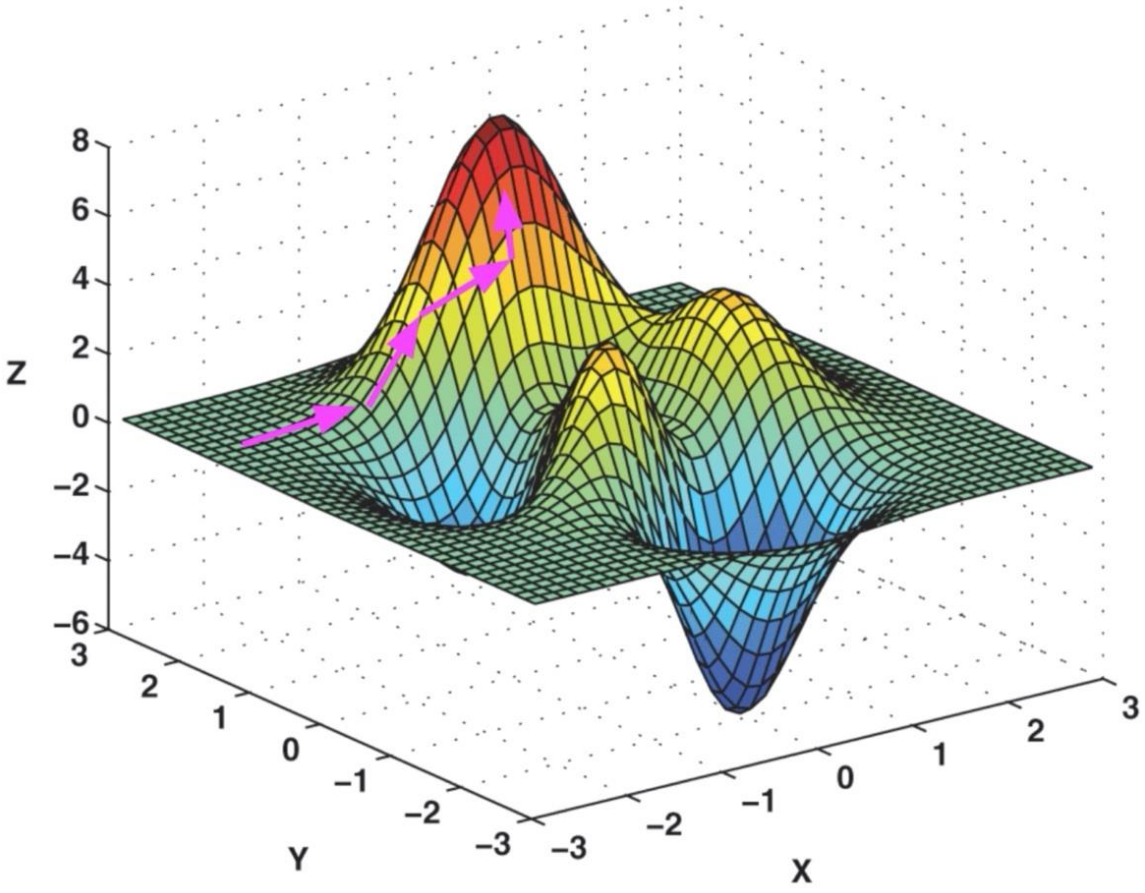
Let’s start with some motivation with biological evolution:

- DNA is a code that describes us (hair color, height, etc.)
- Assume we are dealing with single celled organisms that copy themselves to produce offspring (rather than 2 parent organisms which is more complicated)
- DNA copying isn’t perfect!
- Mutations can be good or bad:
  you can be more athletic or more susceptible to being sick

- But overall the offspring is similar to a parent

Evolution strategy also may be described in the following way. Let's use the optimization example of hill climbing:

Figure 13. Illustration of algorithm



source: udemy.com

Usually, pseudocode for such type of problem looks like:

```
w0 = random point
while not converged:
  w = w0 + noise
  if Fitness(w) > Fitness(w0):
    w0 = w
```

We start at some random point w0. We also going to get the value of the function we want to optimize at this point w0. Let's assume it's somewhere at the bottom of the hill. Then, in the loop, we are going to add some random noise to w0 and call it w. Next, we are going to evaluate our function at the value of w. It could be better or worse than our previous value because the noise we added was random (just like a random mutation in a DNA string). If it's worse – we throw it away but if it's better – we keep it and make it a new value of w0. As we do it iteratively, eventually we are going to get to the top of the hill. The only way we end up keeping a new value of w0 is if we find that it gives us an improvement on our function value. This already sounds a lot like natural selection discussed before but with 1 important point missing: in each generation we don't just have one offspring, we have multiple! But how we update weight w if we have multiple offspring who will perform with the various degree of success? We might say: throw out the bad ones and keep good ones, but how can we combine multiple good offspring? What if 2 offspring are equally good and they have different weights? In fact, having multiple offspring could very useful in reinforcement learning since even with the same policy playing the episode multiple times will yield a different reward each time, therefore aggregating the results of multiple runs from multiple offspring! Possibly playing multiple episode per offspring could be useful in estimating the true expected reward.

We can present the algorithm with the following pseudocode:

```
Given:
  Learning rate η
  Noise standard deviation σ
  Initial policy parameters θ(0)
For t=0...T:
  For n=1...N:
    Sample ε_n ~ N(0, I)
    θ_try = θ(t) + σ * ε_n
    F_n = F(θ_try) # Calculate reward
  # Calculate the new θ
Return θ(T)
```

Scalar Vector

$$\theta(t+1) = \theta(t) + \eta \frac{1}{N\sigma} \sum_{n=1}^{N} F_n \varepsilon_n$$

fig.15 – Illustration of algorithm | source: udemy.com

First, we have to introduce a few more variables:

- Learning rate η (a hyper-parameter that controls how much we are adjusting the weights of our network)
- noise standard deviation σ
- initial policy parameters $\theta(0)$ (theta zero)

Now, for predetermined number of epochs, here is what we do:
We are going to have the inner loop which generates the offspring. Inside this loop we generate some Gaussian noise from the standard normal distribution which represents the offspring. Then, we multiply it by sigma, so that it has a desired standard deviation and add it to our current weight $\theta(t)$ which gives us θ_{try}. Next, we are going to evaluate function F using the current offspring parameters θ_{try}. Note: this function F can be an actual function like a quadratic or it can be the accuracy of supervised/unsupervised learning model or it can be a reward from some environment after playing one or more episodes. That's a great thing about this black box optimization method, it's very flexible, since F can be anything we want to optimize. Net, we update $\theta(t)$ to get $\theta(t+1)$ using the formula above (in green frame). We add a learning rate times 1 over N times σ times the sum of all the rewards multiplied by their corresponding noise vectors.

Now, since we have a strategy for implementation we just need to specify initial parameters.
For my experiment I've chosen population size = 30 (number of offsprings), learning rate = 0.03, sigma = 0.1 and number of iterations = 300. Also, since the library used to obtain environment for flappy bird (PLE) doesn't provide convenient API for controlling reinforcement learning agent, I've implemented all necessary methods in the class of environment and neural network for computational purposes.

Conclusion:

Model with such parameters was training for 15 minutes. Running model with the trained agent afterwards showed that it was able to achieve amazing results. It was continuously flying without falling down for 3 minutes after what I turned it off assuming that it can do it for infinity. For comparison, the average human can do it for 10-15 seconds which tells us about tremendous power of AI.
Talking about possible improvements, there is an option to use multiprocessing in case of more complicated environment which allows us to use several CPUs for a computational process.

Also, if the environment seems to be very computationally demanding, we can always use AWS (Amazon Web Services) and train the agent in cloud.

Appendix.

Code:

```
# Implementation of intelligent agent playing game Flappy Bird
# using deep learning and reinforcement learning (evolution strategy) algorithms

import numpy as np
import matplotlib.pyplot as plt

from datetime import datetime

from ple import PLE
from ple.games.flappybird import FlappyBird

import sys

state
HISTORY_LENGTH = 1


class Env:
  def __init__(self):
    self.game = FlappyBird(pipe_gap=125)
    self.env = PLE(self.game, fps=30, display_screen=False)
    self.env.init()
    self.env.getGameState = self.game.getGameState  # maybe not necessary

    # by convention we want to use (0,1)
    # but the game uses (None, 119)
    self.action_map = self.env.getActionSet()  # [None, 119]
```

```
  def step(self, action):
    action = self.action_map[action]
    reward = self.env.act(action)
    done = self.env.game_over()
    obs = self.get_observation()
    # don't bother returning an info dictionary like gym
    return obs, reward, done

  def reset(self):
    self.env.reset_game()
    return self.get_observation()

  def get_observation(self):
    # game state returns a dictionary which describes
    # the meaning of each value
    # we only want the values
    obs = self.env.getGameState()
    return np.array(list(obs.values()))

  def set_display(self, boolean_value):
    self.env.display_screen = boolean_value


# make a global environment to be used throughout the script
env = Env()


# neural network

# hyperparameters
D = len(env.reset()) * HISTORY_LENGTH  # input size
M = 50  # hidden layer size
K = 2  # output size
```

```
def softmax(a):
  c = np.max(a, axis=1, keepdims=True)
  e = np.exp(a - c)
  return e / e.sum(axis=-1, keepdims=True)


def relu(x):
  return x * (x > 0)


class ANN:
  def __init__(self, D, M, K, f=relu):
    self.D = D
    self.M = M
    self.K = K
    self.f = f

  def init(self):
    D, M, K = self.D, self.M, self.K
    self.W1 = np.random.randn(D, M) / np.sqrt(D)
    # self.W1 = np.zeros((D, M))
    self.b1 = np.zeros(M)
    self.W2 = np.random.randn(M, K) / np.sqrt(M)
    # self.W2 = np.zeros((M, K))
    self.b2 = np.zeros(K)

  # returns a list of probabilities
  def forward(self, X):
    Z = self.f(X.dot(self.W1) + self.b1)
    return softmax(Z.dot(self.W2) + self.b2)

  def sample_action(self, x):
    # assume input is a single state of size (D,)
```

```
    # first make it (N, D) to fit ML conventions
    X = np.atleast_2d(x)
    P = self.forward(X)
    p = P[0]  # the first row
    # return np.random.choice(len(p), p=p)
    return np.argmax(p)

  def get_params(self):
    # return a flat array of parameters
    return np.concatenate([self.W1.flatten(), self.b1, self.W2.flatten(), self.b2])

  # returns a dictionary of all the neural network's weights
  def get_params_dict(self):
    return {
        'W1': self.W1,
        'b1': self.b1,
        'W2': self.W2,
        'b2': self.b2,
    }

  def set_params(self, params):
    # params is a flat list
    # unflatten into individual weights
    D, M, K = self.D, self.M, self.K
    self.W1 = params[:D * M].reshape(D, M)
    self.b1 = params[D * M:D * M + M]
    self.W2 = params[D * M + M:D * M + M + M * K].reshape(M, K)
    self.b2 = params[-K:]


def evolution_strategy(
        f,  # f = reward_function
        population_size,
        sigma,  # standard deviation for the noise
```

```
    lr,
    initial_params,
    num_iters):

# assume initial params is a 1-D array
num_params = len(initial_params)
reward_per_iteration = np.zeros(num_iters)

params = initial_params
for t in range(num_iters):
  t0 = datetime.now()
  N = np.random.randn(population_size, num_params)  # generates the noise

  R = np.zeros(population_size)  # stores the reward

  # loop through each "offspring"
  for j in range(population_size):
    params_try = params + sigma * N[j]  # N[j] = noise for this offspring
    R[j] = f(params_try)

  m = R.mean()
  s = R.std()
  if s == 0:
    # we can't apply the following equation
    print("Skipping")
    continue

  A = (R - m) / s  # standardizing reward
  reward_per_iteration[t] = m
  params = params + lr / (population_size * sigma) * np.dot(N.T, A)

  # update the learning rate
  lr *= 0.992354
  # sigma *= 0.99
```

```
    print("Iter:", t, "Avg Reward: %.3f" % m, "Max:",
          R.max(), "Duration:", (datetime.now() - t0))

  return params, reward_per_iteration

# using a neural network policy to play episode of the game and return the reward


def reward_function(params):
  model = ANN(D, M, K)
  model.set_params(params)

  # play one episode and return the total reward
  episode_reward = 0
  episode_length = 0
  done = False
  obs = env.reset()
  obs_dim = len(obs)
  if HISTORY_LENGTH > 1:
    state = np.zeros(HISTORY_LENGTH * obs_dim)  # current state
    state[-obs_dim:] = obs
  else:
    state = obs
  while not done:
    # get the action
    action = model.sample_action(state)

    # perform the action
    obs, reward, done = env.step(action)

    # update total reward
    episode_reward += reward
    episode_length += 1
```

```
    # update state
    if HISTORY_LENGTH > 1:
      state = np.roll(state, -obs_dim)
      state[-obs_dim:] = obs
    else:
      state = obs
  return episode_reward


if __name__ == '__main__':
  model = ANN(D, M, K)

  if len(sys.argv) > 1 and sys.argv[1] == 'play':
    # play with a saved model
    j = np.load('es_flappy_results.npz')
    best_params = np.concatenate(
        [j['W1'].flatten(), j['b1'], j['W2'].flatten(), j['b2']])

    # in case initial shapes are not correct
    D, M = j['W1'].shape
    K = len(j['b2'])
    model.D, model.M, model.K = D, M, K
  else:
    # train and save
    model.init()
    params = model.get_params()
    best_params, rewards = evolution_strategy(
        f=reward_function,
        population_size=30,
        sigma=0.1,
        lr=0.03,
        initial_params=params,
        num_iters=300,
```

```
  )

  # plot the rewards per iteration
  # plt.plot(rewards)
  # plt.show()
  model.set_params(best_params)
  np.savez(
      'es_flappy_results.npz',
      train=rewards,
      **model.get_params_dict(),
  )

# play 5 test episodes
env.set_display(True)
for _ in range(5):
  print("Test:", reward_function(best_params))
```

Sources:

- https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08
- https://blog.soshace.com/deep-learning-vs-machine-learning-overview-comparison/
- Reinforcement Learning: An Introduction (Richard S. Sutton and Andrew G. Barto, 2014-2015)
- udemy.com
- https://en.wikipedia.org/wiki/Reinforcement_learning
- Reinforcement Learning: Theory and Practice (Csaba Szepesvari)
- https://openai.com/blog/evolution-strategies/
- https://pygame-learning-environment.readthedocs.io/en/latest/
- https://stackoverflow.com/