Annex No. 1 to the Rules for Verifying Independent Preparation and Electronic Archiving of Diploma Theses

Undergraduate/Graduate[1] Programme

Major: Advanced Analytics - Big Data

Minor[2] ————————

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

Bachelor's/Master's[1] thesis written in the Department/Institute[1]
Warsaw School of Economics
Under scientific supervision of
dr. Jarosław Olejniczak

Warsaw 2020

[1] Apply as appropriate

[2] If no major has been selected or no declaration of the major has been filed, skip the entry.

Table of contents

1. Introduction

Objective: to prove that Evolution Strategy algorithm may be used for solving Reinforcement Learning problems and be as sufficient as older, proven to be effective algorithms, such as: Q-learning, SARSA, DQN, DDPG, A3C. The main disadvantage of abovementioned algorithms is that they are highly relying on MDP[1] (Markov Decision Process) framework which is very computationally demanding due to recursion involved for solving its main equation - Bellman Equation[2] (also known as a Value Function). Thus, all these algorithms can't be used efficiently if the one does not have the access to a bunch of GPUs which are usually required in order to train a good agent. With my experiment I'm going to prove that even having 8GB RAM laptop may be enough for RL task using Evolution Strategy algorithm since it does not use Bellman Equation and, hence, not based on recursion which may require tremendous amounts of computational power.
The experiment will be conducted on the example of mobile game "Flappy Bird".
Implementation of the same game using Q-learning algorithm took about 6-7 hours in order to train the agent (the link to this experiment attached in "Literature" section). I claim that with Evolution Strategy a good agent may be trained in a shorter period.

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)[3] based on Reinforcement Learning technology and trained it

[1] Richard S. Sutton & Andrew G. Barto (2014). *Reinforcement Learning: An Introduction*, The MIT Press Cambridge, Massachusetts London, England, p. 67-69

[2] Bellman, R.E. (1957). *Dynamic Programming*. Dover. Accessed: 12 December 2019

[3] "A Google DeepMind Algorithm Uses Deep Learning and More to Master the Game of Go | MIT Technology Review" (2016). *MIT Technology Review*. Accessed: 12 December 2019

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.

2. Introduction to Machine Learning

We have seen Machine Learning as a buzzword for the past few years, it may be explained by extensive increase in the volumes of data by application and increase of computing power in the last few years and development of more advanced algorithms.

Machine learning is used everywhere, from automating routine tasks to intelligent understanding, and industries in every sector are trying to capitalize on this. We already use devices that use it. For example, a fitness tracker, such as Fitbit, or a smart home assistant, such as Google Home. But there are many more examples of using ML

- Prediction — also Machine Learning is used in prediction systems. Considering the loan example, in order to compute a fault's probability, it's needed to classify the data available in groups.
- Image recognition —ML can is also used for detecting faces on images. Each person assigned to a different category in the database.
- Speech Recognition — spoken words translated to text. Used for recognition of voice. Voice user interfaces: call routing, voice dialing, device management. Simple data entry and the preparation of documents can also be used.
- Medical diagnoses — Machine Learning is trained to find out cancerous tissues.
- Trading and financial industry — ML is used in fraud detection and credit checks.

2.1 A Quick History of Machine Learning

It was in the 1940s the first operated manually computer system, ENIAC (Electronic Numerical Integrator and Computer), has been invented[4]. At that time "computer" was used as the name for human with high numerical computation capabilities, thus, ENIAC was named a numerical computing machine! You may think it had nothing in common with learning?! WRONG, from the very beginning idea was to build machine which is able to imitate human learning and thinking.

Figure 1. Electronic Numerical Integrator and Computer

source: www.computerhistory.org. Accessed: 4 December 2019

In the 1950s, we saw that first computer program claimed to be able to win against the checkers world champion. The program has helped checkers players to improve their skills tremendously!

[4] McCartney, Scott (1999). *Eniac: The Triumphs and Tragedies of the World's First Computer*, Walker & Co

The same time, Frank Rosenblat invented the perceptron[5], it was a very simple classifier but combined into large numbers, a network, it has become a powerful machine. The monster was relative to time and that time, that was real breakthrough. After that we saw few years of stagnation neural networks field because of its difficulties solving certain problems.

Thanks statistics, ML became really famous in 1990s. The combination of statistics and computer science gave birth to AI. It has shifted this field further towards data-driven approaches. With large-scale data, scientists have built intelligent systems which were able to learn and analyze huge amounts of data. Deep Blue system has beaten world chess champion, Garry Kasparov. Kasparov accused IBM in cheating, but now it's piece of history and Deep Blue rests peacefully in museum.

2.2 What is Machine Learning?

According to Arthur Samuel, ML algorithms let computers learning from data, and improve themselves, even without being programmed explicitly[6].

Machine learning is category of algorithms that allow software to become more precise in outcomes prediction without being programmed explicitly. The basic feature of ML is to build algorithm that receives input data and with the use of statistical analysis predicts output updating output as a new data available.

2.3 Types of Machine Learning

ML may be classified into 3 algorithm types.

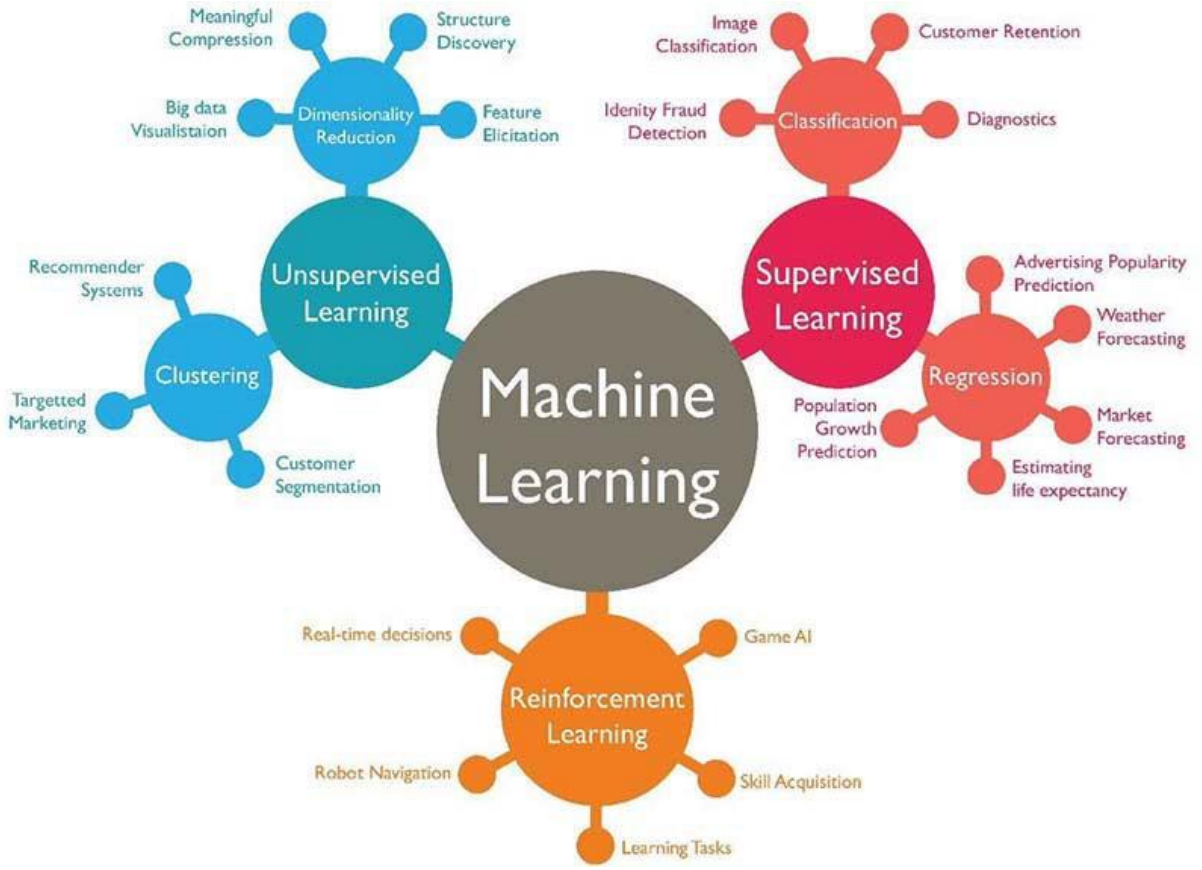
1. Supervised Learning
2. Unsupervised Learning

[5] Rosenblatt, Frank (1962), *Principles of Neurodynamics*. Washington, DC:Spartan Books.

[6] Arthur, Samuel (1959). *Some Studies in Machine Learning Using the Game of Checkers.* IBM Journal of Research and Development. 3, p.210–229. Accessed: 12 December 2019

3. Reinforcement Learning

Figure 2. Types of Learning



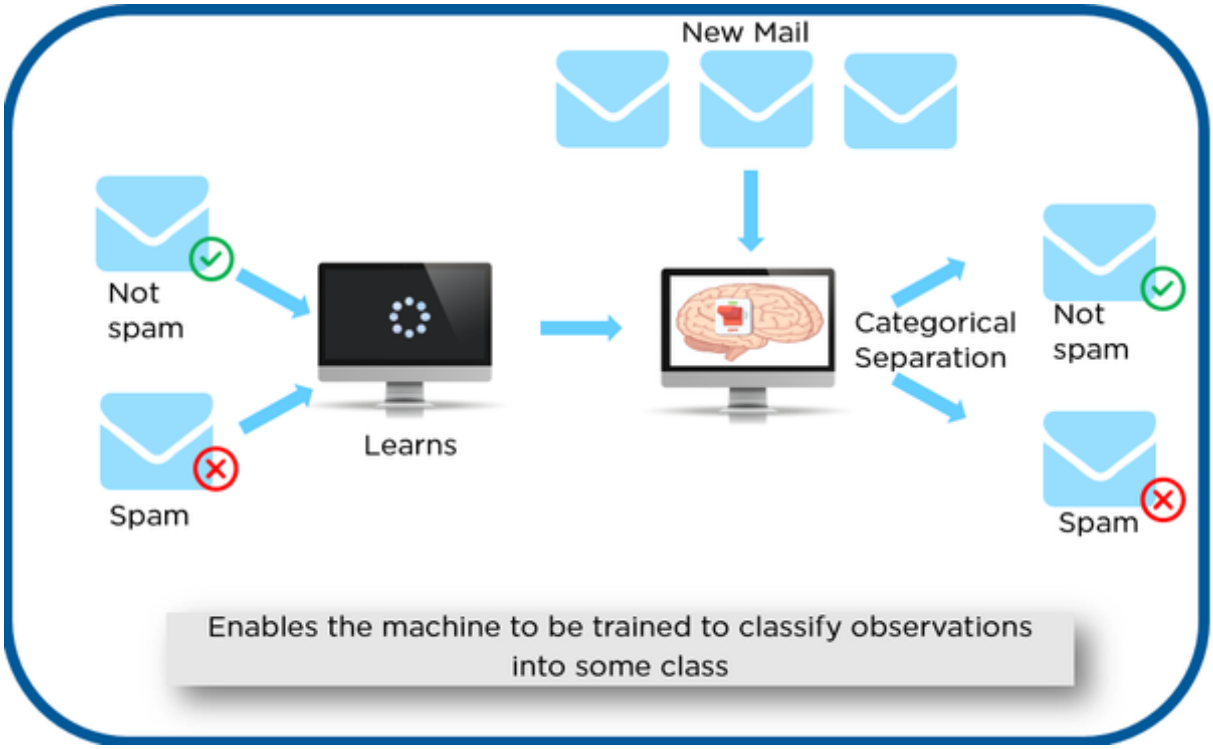
source: towardsdatascience.com. Accessed: 8 December 2019

2.4 Overview of Supervised Learning Algorithm

In Supervised learning, AI system presented with the labelled data, it means that each data marked with the correct label.

The goal is to optimize the mapping function that well so when the new input data (x) is used we can predict the output (Y) of that data.

Figure 3. Example of Supervised Learning



source: towardsdatascince.com. Accessed: 12 December 2019

In the example above some data were initially taken and marked as 'Spam'/'Not Spam'. This data was used for training supervised model.

After it has trained, the model can be tested with new mails and assessing if the model able to predict the correct output.

2.5 Types of Supervised learning

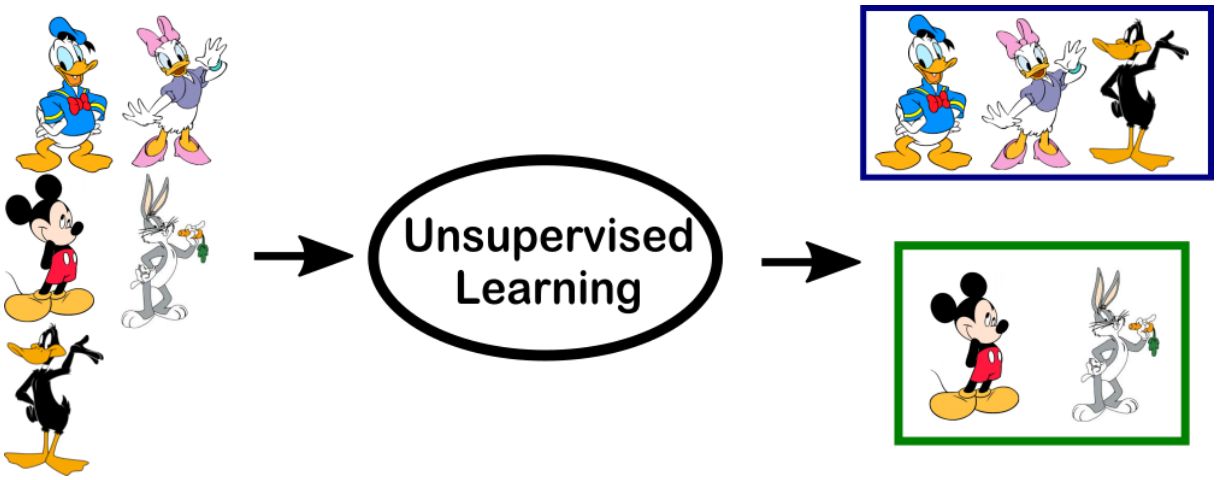
- **Classification**: the output variable is category, such as "blue", "red" or "disease" or "no disease".

- **Regression**: the output variable is real value, like "weight" or "dollars".

2.6 Unsupervised Learning

In unsupervised learning, the AI system presented with uncategorized, unlabeled data and the system algorithm act on data without training beforehand. The output is dependent on the coded algorithm. AI may be tested by subjecting unsupervised learning as a system.

Figure 4. Example of Unsupervised Learning



source: towardsdatascince.com. Accessed: 12 December 2019

In the example above, we gave some characters to the model: 'Ducks'/'Not Ducks'. In the training data, we don't specify any label to the data. The unsupervised model can separate both characters by looking at type of the data and models the structure/distribution in data to learn more about it.

2.7 Types of Unsupervised learning

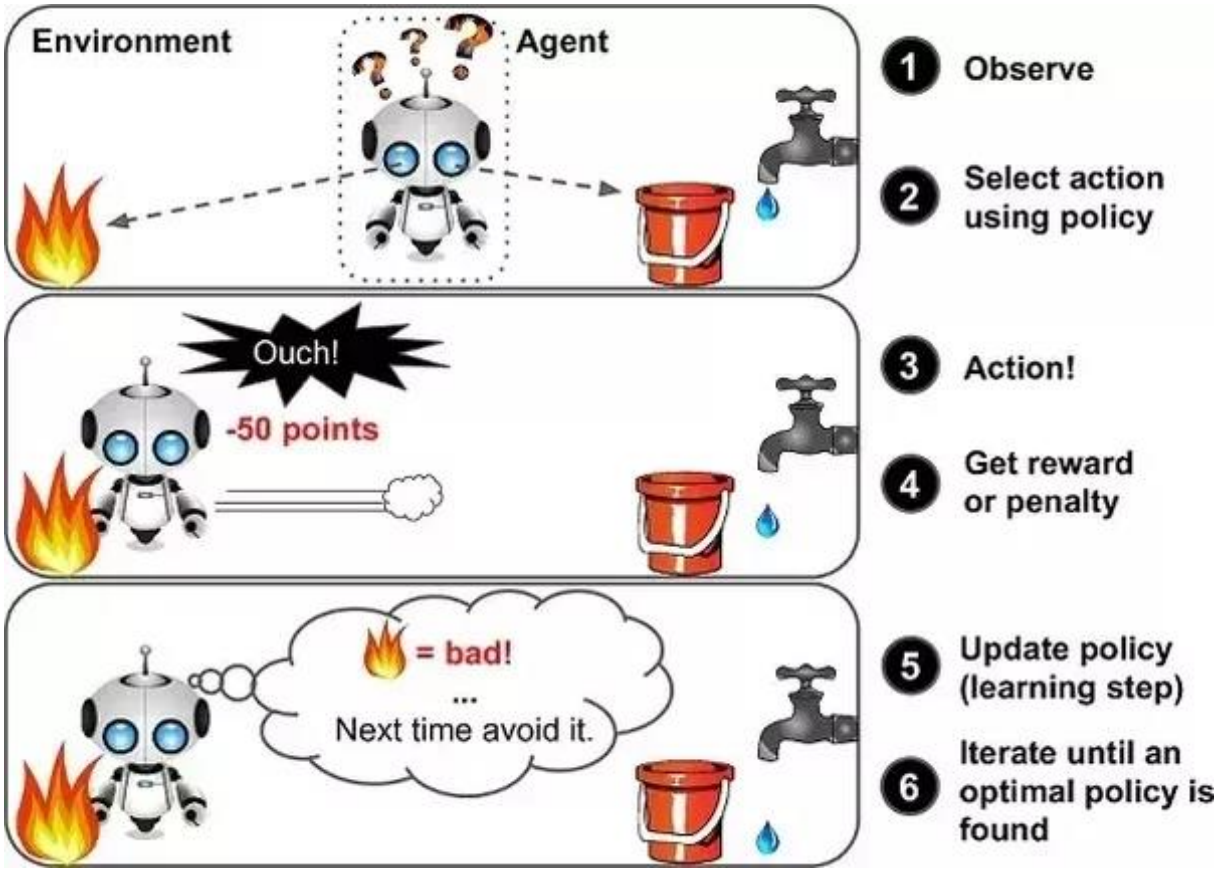
- **Clustering**: problem where you want to find out inherent groupings of the data, like grouping customers by their purchasing behavior.
- **Association**: problem where you want to find out rules that can describe huge portions of the data, like people that bought X are probably going to buy Y.

2.8 Reinforcement Learning

A reinforcement learning algorithm, an agent, learns while interacting with the environment. The agent gets rewards performing correctly and is penalized for bad performance. The agent is

learnt without human intervention by maximizing reward and decreasing its penalty. It is a kind of dynamic programming which trains algorithms with the use of a system: reward and punishment.

Figure 5. Example of Reinforcement Learning



source: towardsdatascince.com. Accessed: 14 December 2019

In the example above, it's seen that agent was given 2 options: a path to water or a path to fire. RL algorithm works on system of reward: if agent uses the path to fire then rewards it gets a negative reward, then agent learns that it has to avoid the path with fire. If it chose the path with water (the safe path) – few points would be added to reward points, then the agent would try learning which path is safe, what isn't. It leverages the rewards obtained, agent improves its knowledge about environment in order to choose the next action.

2.9 Deep learning

Deep learning, is a subset of ML, is inspired by the processing of information patterns similar to the ones in human brain. The brain processes the information, disaggregates it, and separates it into categories. When gets new information, brain compares it to the existing one and makes conclusion that allows to make future action according to the analysis. Deep learning based on many layers of algorithms, each providing different interpretation of data that's been fed to them.

In case of supervised learning, the user has to train the AI so that it makes the right decision: user gives the input to the machine and gets the expected output, in case the output is wrong, it calculates from beginning; this process goes on till the AI is able to make predictions with no mistakes. Popular supervised algorithms: logistic regression, linear regression, decision trees, non-parametric models such as k-Nearest Neighbors[7] and support vector machines. In case of unsupervised learning, user lets AI to make classifications from data. Algorithms such as k-Means, hierarchical clustering, Gaussian models which attempt to learn data structures.

Deep Learning doesn't use any strict rules as the Machine Learning algorithm should be able to extract trends and patterns out of the huge sets of data.

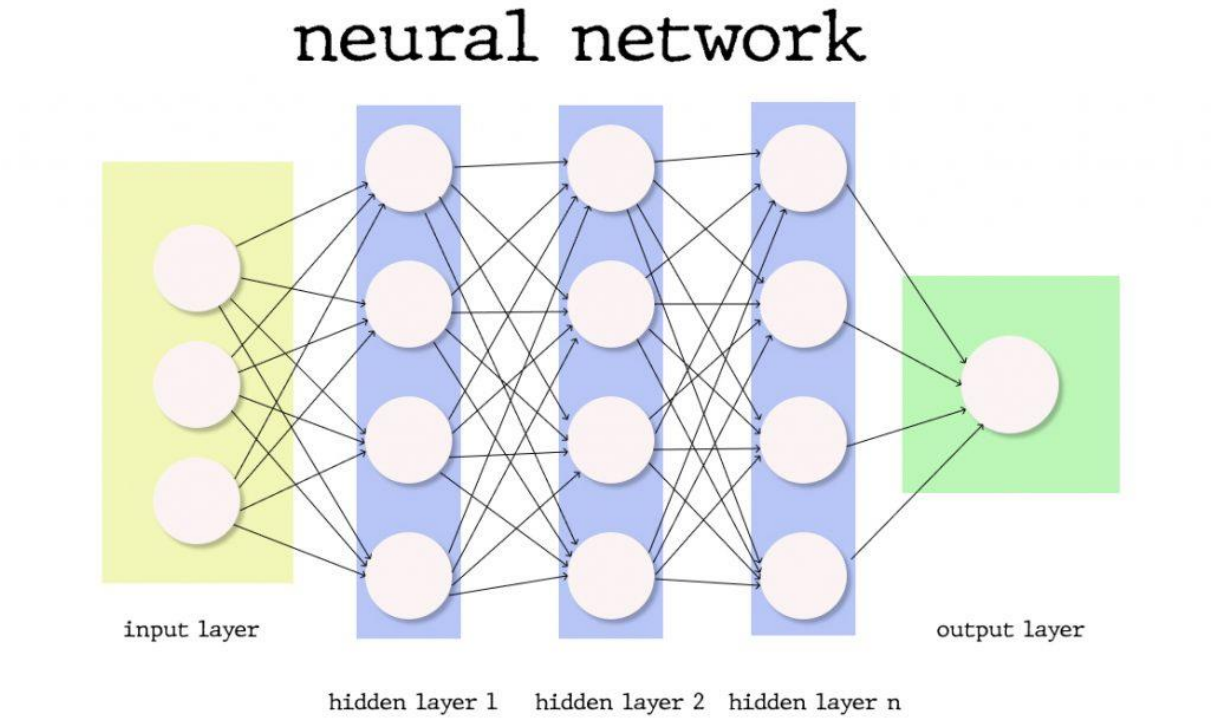
2.10 Neural Network vs Deep Learning

The main algorithm of Deep Learning is neural networks — the more layers it has, the deeper is the network. Each layer contains computational nodes, so-called "neurons," each of them is connected to all neurons in layer. There are few types of layers:

- The input layer, receives information and sends it to underlying nodes
- Hidden layers are those which make all the calculations
- Output node layer – a place for results of computation

[7] Altman, Naomi S. (1992). *An introduction to kernel and nearest-neighbor nonparametric regression.* The American Statistician, p.175–185. Accessed: 18 December 2019

Figure 6. Neural Network



source: towardsdatascince.com. Accessed: 15 December 2019

With addition of more hidden layers to the network, the scientists enable deeper calculations; Although, the more layers there are — the more computing power is needed in order to deploy such network.

Each connection has importance and weight, its initial values are randomly assigned or assigned according to its importance perceived for the Machine Learning model. Activation function[8] for each neuron is evaluated in the way the signal is taken, and if the analyzed data is different from the expected, weight values are generated again, and the iteration begins. The difference between expected results and the yielded ones called a loss function, it should be as close as possible to zero. Gradient Descent[9] - function describing how the change in connection importance affects the accuracy output. After each of iterations, we adjust nodes' weights into

[8] Elliot, David L. (1993), *A better activation function for artificial neural networks*

[9] Barzilai, Jonathan; Borwein, Jonathan M. (1988). *Two-Point Step Size Gradient Methods*. IMA Journal of Numerical Analysis. P.141–148. Accessed: 18 December 2019

small increments in order to find out direction for reaching the set minimum. After n number of iterations, trained DL model is supposed to produce accurate results and may be deployed into production, although, some adjustments may be necessary in case the weight of factors changes over the time[10].

2.11 Summary of how Deep Learning works

DL is one of ways to implement Machine Learning with artificial neural networks, such algorithms that replicate the human's brain structure. In fact, Deep Learning algorithms use many layers in order to progressively extract useful features from the input[11]. In Deep Learning, levels learn to transform the input data to abstract representation, a deep learning networks can easily learn which features to place on which level, without human interaction. Deep Learning is both applied for unsupervised and supervised learning tasks. For supervised tasks Deep Learning methods don't require feature engineering and extract layered structures removing redundancy in presentation; Deep Learning structures that are used in unsupervised manner are neural history compressors and deep belief networks.

2.12 Application of Deep Learning

Let's have a look at some top applications of DL, which may help you to understand DL better and the way it works.

The most known deep learning application is a recommendation engine[12] which enhances the user experience, providing a better user service. Two types of recommendation engines: collaborative filtering and content-based. Unless you have sizable user-base, it is best to start with the content-based engine first.

NLP and RNN are used in the text in order to extract higher-level information, known as a text sentiment analysis.

---

[10] Martin Anthony (January 2001). *Discrete Mathematics of Neural Networks: Selected Topics*. SIAM. pp. 3

[11] Haykin, Simon S. (1999). *Neural Networks: A Comprehensive Foundation*

[12] Adomavicius, G.; Tuzhilin, A. (June 2005). *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions"*

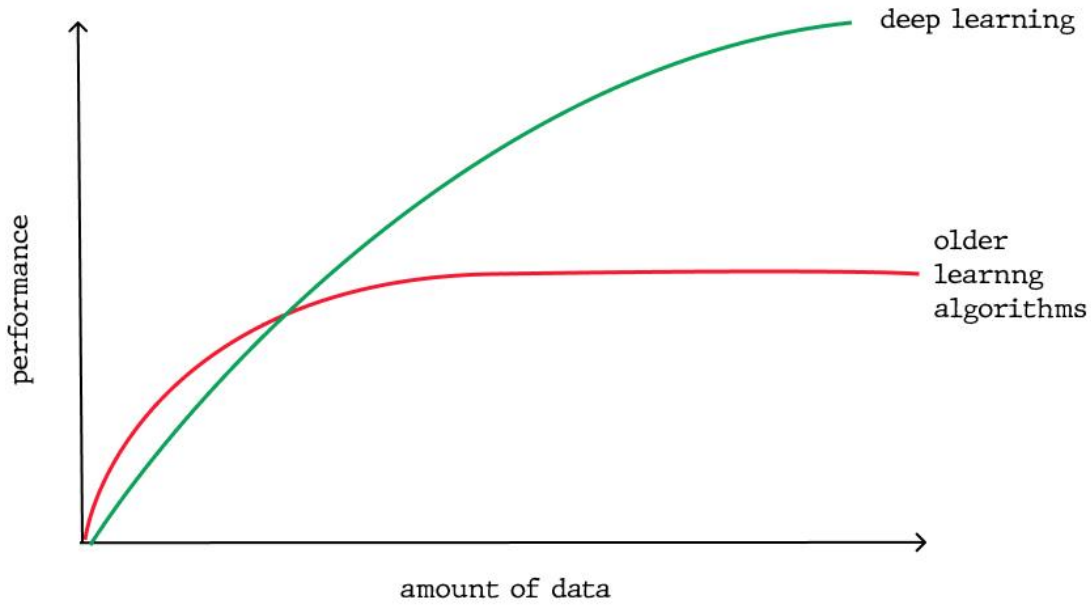
Another type of application is chatbots which may be trained with multiple samples of dialogs and RNN.

Another type of application of Deep Learning models is image classification[13] using recognition models in order to sort images to different sub-categories or to use auto-encoders in order to retrieve images basing on their visual similarity.

2.13 Comparison of Machine Learning and Deep Learning

The most notable differences is in scalability of DL versus older ML algorithms: when amount of data is small, DL doesn't perform well, but when amount of data increases, DL skyrockets in performing on that data; traditional algorithms don't scale with the amount of data as much.

Figure 7. Scaling with Amount of Data



source: towardsdatascince.com. Accessed: 16 December 2019

Another important difference which comes directly from first difference is the DL hardware dependency: Deep Learning algorithms depend on GPUs and high-end machines, they are

[13] Richard Szeliski (2010). *Computer Vision: Algorithms and Applications*

doing large number of matrix multiplications, whereas the older ML algorithms can work on weaker machines absolutely well.

In ML, the most of applied features have to be identified with a help of ML expert, who then copies them per domain and type of data. The input values may be anything from shapes, pixel values, textures, etc. Performance of older ML algorithm will depend significantly on how accurately the features have been inputted, extracted, identified[14]. DL learns high-level features from the data, it is a major difference from ML because it decreases the task: developing new features extractor for the problem, in turn, DL learns lol-level features in the layers of neural network and high-level when it goes deeper to the network.

Because of tremendous amount of data it needs to learn from, DL algorithms take lot of time in order to train, it can even be several weeks sometimes. To compare, ML takes less time for training: from few seconds to few hours. However, while testing, DL takes less time to be run than the average ML algorithm.

Also, interpretability is another factor to compare. With DL algorithms, it may be impossible to interpret results, it's exactly the reason why some of industries had slow adaptation of DL. Nevertheless, Deep Learning models still may achieve high precision but with the cost of bigger abstraction. To extend this topic more, let's back to weights of neural network, which indicates a measure of strength of connection between neurons. So, looking on first layer, we may say what is the strength of connection between first layer of neurons and the input. However, at second level, we can easily lose the relationship, since one-to-many has turned to many-to-many relationship, it's explained by a high complexity on NN: neuron in one layer may be related to other neurons that are far from first layer, deeper into network. Weights tell story about input, although the information compressed after applying the activation functions to make it almost impossible to decode. On the other hand, ML algorithms are like decision trees, the give certain rules why it choose what it chose, hence, they are much easier to interpret.

[14] (2015). *"Feature Engineering: How to transform variables and create new ones?"*. Analytics Vidhya. Accessed: 18 December 2019

3. Theory behind Reinforcement Learning

The idea to learn by interacting with environment is probably first which occurs to us when we think of nature of learning[15]. When the infant plays, looks about or waves its arms, it has no teacher, but what it has is the explicit sensorimotor connection with its environment. Exercising the connection yields a wealth information about the cause and the effect, about what to do to achieve the goal and the consequence of the action. During our lives, such interactions are a major source of the knowledge about the environment and about ourselves. Whether we are learning to maintain a conversation or to drive a car, we are absolutely aware of how the environment will respond to our actions, and we strive to influence what will happen using our behaviour. Learning through interaction is foundational concept which lies under almost all learning theories. Instead of directly making theory about people and animals learning, we explore imitated learning situations evaluating the effectiveness of different learning methods. Thus, adopting perspective of AI engineer or researcher. Exploring designs of machines that effective problem-solving of economic or scientific interest, evaluating designs with computational experiments or mathematical analysis. The explored approach is called reinforcement learning, it is more focused on directed learning by interacting rather than other approaches to machine learning[16].

RL problems' purpose is to learn what to do and how to align situations with actions so it allows to maximize reward signal. They are basically a closed-loop problem since learning system's action affects its later inputs. The learner doesn't get any instructions about which actions to choose, as in other many forms of ML, instead it has to discover which actions may bring the most reward simply trying each of them. In most challenging and interesting cases, these actions may change not only immediate reward, also the following situation and, with that, all consequent rewards. These characteristics are closed-loop in a certain way, not having instructions of what actions to take, where consequences of these actions, including reward, lead over some time periods are three important features of RL problems. Full specification of RL problems with the optimal control are described by Markov decision processes, it is the idea of capturing the most important features of real problem for a learning agent which interacts

[15] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*. p.237–285. Accessed: 22 December 2019

[16] Watkins, Christopher J.C.H. (1989). *Learning from Delayed Rewards*. King's College, Cambridge, UK.

with the environment in order to achieve the goal. Such agent have to be able to feel the state of environment to some degree and should be able of taking actions which affect the state. The agent must as well have goals related to the state of environment. Generally, three aspects are included: action, goal and sensation in the simplest forms without generalizing any of them. Any good method which is suitable to solve this type of problem is considered to be a RL method. RL differs a lot from supervised learning, the type of learning which is studied in the most current researches in the field of ML. RL also differs from what ML researchers name unsupervised learning, which is usually all about looking for a structure hidden in the unlabelled data collections.

The terms unsupervised learning[17] and supervised learning appear to classify ML paradigms, but they actually do not. Although one might think of RL as a type of unsupervised learning since it is not relying on the examples of correct behaviour, RL tries to maximize the reward instead of looking for a hidden structure. Uncovering structure in the agent's experience may certainly be used in RL, but by isn't addressing the RL problem – to maximize a reward. Hence, RL is considered to be a third ML paradigm, alongside of unsupervised learning, supervised learning[18] and may be other paradigms. One of the biggest challenges arising in RL, which is not present in the other types of ML, is a trade-off between exploitation and exploration. To get a bigger reward, a RL agent has to prefer actions that have been tried in past and still be effective in yielding rewards. To discover this type of actions, it must try some actions that haven't been selected before. The agent must exploit what it knows already to get the reward, but at the same time it must explore – making better action choices in future. The dilemma is: neither exploitation nor exploration may be pursued without a risk to fail at task. The agent has to try a variety of different actions and iteratively choose those that seems to be the best ones. On stochastic task, every action has to be tried multiple times in order to gain a decent estimate - expected reward. The exploitation-exploration dilemma was intensively studied by different mathematicians for many years.

Another main feature of RL is that it considers explicitly the whole problem of agent as a goal-directed interacting in an uncertain environment. This is in opposite to many approaches that look at subproblems without thinking how they might possibly get to a larger picture. For

[17] Hinton, Geoffrey; Sejnowski, Terrence (1999). *Unsupervised Learning: Foundations of Neural Computation*. MIT Press.
[18] Vapnik, V. N. *The Nature of Statistical Learning Theory*, Springer Verlag, 2000.

instance, it was mentioned that most of ML research is connected with supervised learning without specifying how such possibility may be useful. Other researchers developed theories: planning with main goals, without considering a role of planning in the decision-making real-time, or question: where predictive models are necessary in order to plan would come from. Although these approaches yielded a lot of useful results, the focus on the subproblems is a tremendous limitation. RL takes the opposite approach, it starts with a goal-seeking, interactive and complete agent. All RL agents have concrete goals, can feel the aspects of environments, and are able to choose proper actions in order to affect their environments. Moreover, usually it's assumed from the very beginning that agent must operate in spite of significant environment's uncertainty it faces. When RL involves planning, it must address the interplay between real-time action selection and planning, as well as question of how models of environment are acquired and improved. To make progress for learning research, subproblems must be studied and isolated, however they should represent subproblems playing clear roles in interactive, complete and goal-oriented agents, even if all of the details of complete agent cannot be filled in yet. One of most exciting features of modern RL is its fruitful and substantive interactions with other scientific and engineering disciplines. RL is part of long trend within AI and ML towards greater integration with optimization, statistics and other computing subjects. For instance, ability of some RL methods to learn with several parameterized approximators demonstrates the classical "dimensionality curse" in control theory and operations research. Moreover, RL also has interacted strongly with neuroscience and psychology, with significant benefits which go both ways. Amongst all forms of ML, RL is the closest to such kind of learning process that humans and the other animals do; many of core algorithms of RL were initially inspired by the biological learning systems. And RL also gave back, both through psychological model of animal's learning and through the influential model of certain parts of brain's reward system. Finally, RL is also a part of the larger trend in AI back towards simple general principles. Since late 1960's, a lot of AI researchers assumed that there are no common principles to discover. Sometimes it was said that if we could get enough facts into machine, let' say one billion, it would become an intelligent one. Methods are based on the general principles: search or learning, are known as "weak methods", those which are based on specific certain knowledge are called "strong methods." Such view is still known today, but less dominant. Modern AI includes much more research which looks for general learning principles, decision-making and search, trying to combine the vast amounts of domain knowledge. Now it is not clear enough how far the pendulum is going to swing, but RL research is definitely a part of a swing back towards the fewer and simpler common principles of AI.

3.1 Examples of application of reinforcement learning

A good way to understand reinforcement learning is to take a look at some of examples and applications that guided its development.

- A chess master makes a move. The choice informed by both: planning/predicting possible answers and counterreplies by intuitive, immediate judgments of desirability of particular moves and positions.
- Adaptive controller is adjusting parameters of petroleum's refinery' operation in a real time. Controller optimizes the cost/yield/quality trade-off based on the specified marginal costs with no sticking strictly to set points which are originally suggested by the engineers.
- Gazelle calf struggles to feet few minutes after it's born. A half an hour later it runs 20 mph.
- Mobile robot makes a decision about whether it has to enter new room in the search of more trash which it has to collect or to start trying to look for its way back towards the battery recharging station. Robot makes decision based on current level of charge of the battery and how easily and quickly it can find its recharger in the past.
- Phil is preparing his breakfast. When it's examined closely, even though this mundane activity requires a complex sequence of behaviour and goal-subgoal relationships: walking the cupboard, then opening it, selecting cereal box, reaching for and retrieving the box. A lot of other tuned, complex, interactive behavioural sequences are needed to get a spoon, bowl and milk jug. At each step series of eye movements is involved to get the information and guide reaching and locomotion. Quick judgments are made continually about how to carry different objects or is it better to put some of them onto the dining table. At each step, guided by goals: to grasp a spoon or to get to refrigerator, and also is in service for other goals: having the spoon which can be used to eat with once the cereal is made and obtaining nourishment. Despite the fact he knows it or not, Phil accesses the information about state of the body which determines nutritional needs, food preferences and level of hunger. Such examples share some features that are that basic, so they are very easy to overlook. They all involve significant interaction between the active environment and decision-making agent, where agent strives to achieve the goal in spite of uncertainty about the environment. The actions of the agent are permitted to affect future state of environment (next chess position, level of refinery's reservoirs, the next location of the robot and future charge level of the battery), therefore affecting options and opportunities which are available to agent at the later times. The optimal choice requires to take into the account delayed, indirect consequences of the actions; thus, it may require planning or

foresight. Also, in these examples effects of the actions can't be predicted fully; thus, the agent has to monitor the environment very frequently and hence react appropriately. For instance, Phil must watch milk he pours to cereal bowl in order to avoid overflowing. All examples involve goals which are quite explicit in sense that agent judges progress towards the goal based on what can be sensed directly. The chess player almost always knows whether he is going to win or not, refinery controller - how much petroleum was produced, mobile robot is aware of when the batteries are going to run down, Phil knows if he is going to enjoy his breakfast. Neither the environment nor the agent may know what we usually think of as the agent and the environment. The agent isn't necessarily the entire organism or robot, and the environment isn't necessarily only what outside of organism or robot. Example of robot's battery is a part of environment of the controlling agent, Phil's degree food and hunger preferences are the features of its environment of the internal agent which makes decision. State of the agent's environment usually includes the information about state of organism or machine in which agent resides, and it may include the memories and even the aspirations.

In all these examples agent may use the experience in order to improve the performance over the time. The chess player uses the intuition which he uses in order to evaluate positions, therefore improving his play; gazelle calf improves efficiency it can run; whereas Phil is learning to streamline of making the breakfast. Knowledge the agent is bringing to task at start either from built to it by the design/evolution or previous experience: related tasks are the influences what is easy or useful to learn, whereas the interaction with environment is what really essential to adjust behaviour in order to exploit certain features of task.

3.2 Elements of RL

Besides the agent and environment, few main sub elements of RL system may be identified: a reward signal[19], a policy, a value function[20] and a model of environment. Policy is something which defines the way the agent learns and behaves in a given environment. Strictly speaking, policy is a connection from perceived states of environment to the actions taken in those states. It the same what in the psychology called: set of associations or rules. In some cases, policy can be a simple lookup table or function, whereas in the others it can involve huge computation

[19] Kaplan, F. and Oudeyer, P. (2004). *Maximizing learning progress: an internal reward system for development*. Embodied artificial intelligence, p.629. Accessed: 28 December 2019

[20] Clarke, Frank H.; Loewen, Philip D. (1986). *"The Value Function in Optimal Control: Sensitivity, Controllability, and Time-Optimality"*. SIAM Journal on Control and Optimization. p.243–263

as a process of searching. Policy is core of the RL agent in sense that alone is quite sufficient to figure out its behaviour. Generally, policies can be stochastic.[21] The reward signal defines goal in a RL problem. Once at each time step, environment is sending to RL agent a number, which represents a reward. Agent's objective is maximizing total reward which it receives in the long run. Reward signal defines what is good and what is bad event for agent. In the biological system, we usually think about rewards as the analogous to experiences of pain and pleasure. They are immediate and define the features of problem the agent faces. Reward sent to agent in any time is depending on current action and current state of environment the agent is currently in. Agent can't change the process doing it. Only way for the agent to influence reward signal - through the actions, which may have the direct effect on the reward, or the indirect effect changing the state of environment. In the example above: Phil eats a breakfast, the RL agent directs his behaviour receiving different signals of reward when he eats the breakfast according to what mood he has, how hungry he is and other sensors of his body, which is the part of the internal RL agent's environment. Reward signal is primary basics of altering policy. If the action which is selected by policy followed by the low reward, then policy can be switched selecting another action in such situation in future. Generally, signals of reward can be a stochastic function[22] of state of environment and scope of actions to be taken. Whereas reward signal tells what's good in immediate sense, the value function tells what's good in long run. Strictly speaking, value of the state is total number of rewards the agent may expect to get over the future, if starts from the state. Rewards determine immediate desirability of the environmental states, values tell about the long-term goodness of the states after states they are likely to follow are taken into account, and rewards are available in these states. For instance, the state may usually yield the low immediate reward while still having a big value because of it regularly followed by the other states yielding high rewards. Reverse may be true as well. Making a human analogy, the rewards are something like pleasure (if it's high) and the pain (if it's low), whereas the values are corresponding to the more refined judgment of how strongly displeased or pleased we are so that the environment is in the particular state. When expressed this way, it's quite clear that the value functions are formalizing the basic and familiar idea. The rewards are in the sense primary, values as predicted outcome of rewards - secondary. Without any rewards there can't be any values, the only sense of estimating values - achieve more reward. However, these are the values with which we mostly concerned making and

[21] Hartmann, A.K. (2009). *Practical Guide to Computer Simulations*. World Scientific.

[22] Joseph L. Doob (1990). *Stochastic processes*. Wiley. P.46-47. Accessed: 28 December 2019

evaluating our decisions. The choice of actions is made based on our value judgments. We are looking for actions bringing about states of the highest value, not the highest reward, since these actions contain the highest amount of reward in the long run[23]. In planning and decision-making, the quantity which is called value - the one which we are most concerned. It is harder to find the values than to determine the rewards. Rewards are given by environment, but the values have to be estimated from sequences of the observations the agent makes in its lifetime. In fact, the most valuable component of all RL algorithms is the method of efficient estimation of values. Central role of the value estimation[24] is the most important concept we learned about RL in the last decades. The fourth, final element of RL systems is the model of environment. It's something that replicates the behaviour of environment, more generally, something that allows conclusions to be made of how the environment behaves. For instance, given an action and state, model can predict the next state and reward. Such models are commonly used for the planning, by it we mean any possible way to decide on course of the actions, bearing in mind all possible future cases before they are experienced. Such methods of solving RL problems using models and planning called methods of model-based, opposed to a simpler free-model methods which are explicitly error-and-trial learners which are viewed as the opposite of the planning.

3.3 Scope and Limitations of RL

Most of RL methods structured around estimation of value functions, although it's not strictly necessary doing this in order to solve RL problems. For instance, some methods such as a simulated annealing, genetic algorithms, genetic programming and other optimization methods are used to approach RL problems without appealing to the value functions. Such methods are evaluating the life-time behaviour for plenty non-learning agents, where each uses the different policy for interaction with the environment and selects those which are able to get the best reward. Such methods are called evolutionary methods since their operation is similar to way the biological evolution creates organisms with the skilled behaviour when they don't learn through the individual lifetimes. If space of the policies is small enough, or may be structured in the way that good policies are common and easy to find, if lot of time available for search

[23] L. C. G. Rogers; David Williams (2000). *Diffusions, Markov Processes, and Martingales: Volume 1, Foundations*. Cambridge University Press. p.121–124. Accessed: 4 January 2020

[24] Zhou, X. Y. (1990). "Maximum Principle, Dynamic Programming, and their Connection in Deterministic Control". *Journal of Optimization Theory and Applications*. p.363–373

then such evolutionary methods may be very effective. As addition, evolutionary methods possess advantages on the problems where learning agent can't accurately feel the state of the environment. Such methods can ignore a lot of useful structure of RL problem: they don't use the fact that policy they're searching for is the function from the states to actions; they don't recognise which states an agent passes over the lifetime, which actions are selected. In some situations, such information may be misleading (when the states are misperceived), but often it can enable even more efficient search. Nevertheless, there are still some methods which, as evolutionary methods, don't appeal to the value functions. Such methods are searching in the spaces of the policies[25] which are defined by the collection of numeric parameters. They estimate directions of the parameters which should be adjusted to quickly improve a performance of the policy. Unlike evolutionary methods[26] they produce such estimates when agent interacts with the environment so that it can take advantage of details from individual behavioural interactions. Such methods are called policy gradient methods, they have proven to be useful in a many problems, some of simplest RL methods fall to this category. Generally, some of such methods are taking advantage of the value function estimates in order to improve the gradient estimates. All in all, distinction between the policy gradient methods[27] and the other methods are included as RL methods are not defined sharply. RL's connection with optimization methods requires additional comment since it's a source of common misunderstanding. If we say a RL agent's goal - to maximize numerical signal of reward, we are not insisting of course that such agent must actually achieve a goal of the maximum reward. When trying to increase the quantity doesn't mean that the quantity will be maximized ever. The point here is that the RL agent is usually trying to maximize the number of rewards it gets. A lot of factors may prevent it from getting the maximum, even though the one may actually exist. Optimality isn't the same as Optimization.

3.4 The Agent-Environment Interface

The RL problem appears to be straightforward framing of problem: learning from the interaction to achieve the goal. The learner and the decision-maker are called agent. The thing

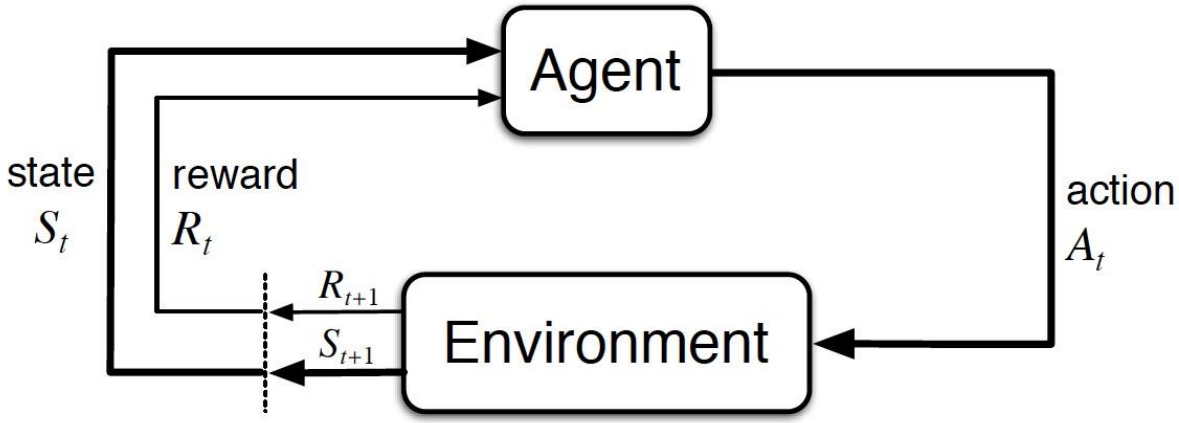
[25] Burnetas, Apostolos N.; Katehakis, Michael N. (1997), *"Optimal adaptive policies for Markov Decision Processes"*, Mathematics of Operations Research, p.222–255. Accessed: 4 January 2020

[26] Ashlock, D. (2006), *Evolutionary Computation for Modeling and Optimization*, Springer. Accessed: 4 January 2020

[27] Elijah Polak (1997). *Optimization : Algorithms and Consistent Approximations*. Springer-Verlag. Accessed: 4 January 2020

it interacts with, comprising everything what is outside of the agent, is called the environment. They are interacting continually; the agent selects the actions and environment responds to these actions presenting the new situations to agent. The environment also

Figure 8. The agent-environment interaction in reinforcement learning



source: Reinforcement Learning: Theory and Practice (Csaba Szepesvari) . Accessed: 22 December 2019

gives rise to rewards, special numeric values which agent is trying to maximize over the time. Complete specification of the environment which defines the task, is one instance of the RL problem. More specifically, agent and the environment are interacting at each of sequence of the 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. A time step later, the part as consequences of the action, an agent receives the numeric reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. With each time step, an agent is implementing the mapping from the states to a probability, selecting each of the possible actions. 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 is changing the policy as the result of the experience. Agent's goal is maximizing the total number of rewards it receives in the long run. This framework is abstract and flexible and can be applied to many different problems in many different ways. For example, time steps don't need to refer to the fixed intervals of the real time; they can refer to arbitrary successive stages of decision-making and acting. Actions may be a low-level control, like voltages which are applied to motors of the robot arm, or the high-level decisions like whether or not having lunch or going to a graduate school. Similarly,

states may take wide variety of different forms. Such states can be determined completely by a low-level sensation like the direct sensor, they may also be a more abstract and high-level like symbolic description of the object in the room.

Some of what makes up a state could be based on memory of previous sensations or to be absolutely subjective or mental. For instance, the agent could be in the state of not being sure where an object is, or having been surprised with a clearly defined sense. Similar to that, some of such actions can be totally computational or mental. For instance, some of actions may control what the agent is choosing to think of, or where it's going to focus the attention. Generally, actions may be any type of decision[28] we would like to learn about how to make, the states may be anything we can know that might be useful in making them. Especially, a boundary between the environment and agent isn't often the same like a physical boundary of the animal's or robot's body. The boundary is usually drawn a bit closer to an agent than that. For instance, the mechanical linkages and motors of the robot and the sensing hardware should be usually considered as parts of environment rather than the parts of agent. If such framework is applied to a person or an animal, the skeleton, muscles and the sensory organs must be considered as a part of environment. Rewards, presumably being computed inside a physical body: artificial and natural learning systems, but they are considered as external to agent. The general rule to follow: anything that can't be changed arbitrarily should be considered outside of it and hence the part of its environment. We aren't assuming that everything in environment is unknown to agent. For instance, the agent usually knows not a lot about the way its rewards computed as function of the actions and states where they were taken. However, reward computation is always considered to be the external to agent because it regulates task of facing the agent, hence has to be beyond the ability to arbitrarily change. Generally, in many cases the agent can know all about how the environment is working but still face a hard RL task as we can know how exactly the Rubik's cube puzzle works, but still being unable solving. The boundary of agent-environment presents the limit of agent's control, but not of the knowledge.

The boundary of agent-environment may be located at a different places with different purposes. For a complicated robot, with many different agents which ca be operating at once, with each having its own boundary. For instance, the agent can make a high-level decision that form part of states which are faced by the lower-level agent which implements a high-level

[28] MacCrimmon, Kenneth R (1968). *"Descriptive and normative implications of the decision-theory postulates."* Palgrave Macmillan, London.

decision. In practice, the agent-environment boundary is determined once one has selected particular states, rewards and actions, and hence identified a certain task of decision-making. The RL framework is the significant abstraction of problem for goal-directed learning by interaction. It may propose that whatever details of memory, sensory and control apparatus, whatever objective is perceived, any type of a goal-directed learning behaviour may be reduced into three signals which are passing back and forth from the agent to the environment: one signal represents choices which are made by agent (so-called actions), one signal representing basis on the choices made (the so-called states), one signal defining the agent's goal (so-called rewards). Such framework can't be sufficient enough to present all decision-learning issues usefully, but it has proven to be useful and applicable widely. For sure, some particular actions and states may vary significantly from one task to another, how they represented may strongly affect the performance. In RL like in the other type of learning, this representational choices are presenting more the art than the science.

3.5 Rewards and Goals

In RL, the goal or the purpose of agent is described in terms of special signal of reward which is passed from environment to agent. With each time step, the reward is simple number, $R_t \in \mathbb{R}$. Formally, agent's goal is maximizing the total number of rewards it will receive. It means to maximize not the immediate reward, but the cumulative one over the long run. Now we can clearly define the informal idea – the hypothesis of reward:

> That all what is meant by the goals and the purposes may be thought of as maximization of expected value of received scalar[29] signal (which is called reward).

Usage of the reward signal[30] in order to formalize an idea of the goal is, perhaps, one of most specific features of RL. Although formulating goals in terms of reward signals might at first appear limiting, it's proven to be very flexible and quite applicable in practice. Another way to look at this is to think of examples how it's been, or may be, used. For example, to make a robot learn to walk[31], researchers provided the reward on the each time step which is proportional to

[29] Brans, C. H. (2005). *"The Roots of scalar-tensor theory"*. Accessed: 4 January 2020

[30] Berridge KC, Kringelbach ML (1 June 2013). *"Neuroscience of affect: brain mechanisms of pleasure and displeasure"*

[31] "Robot density rises globally" (2018). Robotic Industries Association.

robot's forward motion. Making the robot to learn how to get out of the maze, the reward is often -1 for every time step that passes prior to escape; it encourages an agent escaping as fast as possible. Making the robot to learn finding and collecting empty cans of soda for recycling, it might give the reward zero most of time, then the reward of +1 paid for each collected can. It might be also wanted to give a robot some negative rewards if it bumps to things/when someone is yelling at it. For the agent to learn playing chess or checkers, natural rewards +1 for winning, but -1 - losing, and 0 - drawing or all other nonterminal positions. It's clearly seen what is happening in all these examples. An agent usually learns maximizing the reward. In order to make it do something we need, we have to provide rewards in such way so that in maximizing them agent will achieve the goals we set. Thus, it is crucial that rewards we have set up really indicate what has to accomplished. Especially, the signal of rewards is not the best place to impart the agent a prior knowledge of achieving what we would like it to get. For instance, chess-playing agent has to be rewarded only in case of actually winning, and not in case of achieving sub goal as taking the opponent's pieces/gaining control of centre of board. In case there would be a reward for achieving such sub goals, then agent may find the way of achieving them without actually achieving a real goal. For instance, it may find the way of taking opponent's pieces even at cost of the game to be lost. The signal of reward is the way to communicate with the robot what we want to achieve, instead of how you want to achieve it.

Newcomers of RL are a bit surprised sometimes, the rewards defining the learning's goal are computed in environment not in agent. For sure, most animals' goals are recognized by the computations which are occurring inside the animals' bodies, for instance, by food recognizing sensors, pain, hunger and pleasure. However, as discussed in previous section, it can be redrawn, the interface of agent-environment in a way that parts of body would be considered outside of agent. For instance, if goal concerns the internal energy reservoirs of the robot, then they are considered as a part of environment; if goal concerns positions of robot's limbs, then these are considered as a part of environment that is, the boundary of agent is drawn at interface between the their control systems and limb. Such things are internal to robot but they are external to learning agent. It is pretty convenient - place boundary of a learning agent at limit of control instead of limit of the physical body. The reason for doing this is that the goal of the agent has to be something for which it has not a perfect control, for example, it must not be able, to decree that reward was received in such a way that it can change the action. Thus, we are placing the source of reward outside of agent. It doesn't preclude the agent of defining for

him some kind of internal reward/a sequence of rewards. Indeed, this is exactly what many reinforcement learning methods do.

3.6 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[32] 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 which precise aspect of the sequence we want 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:

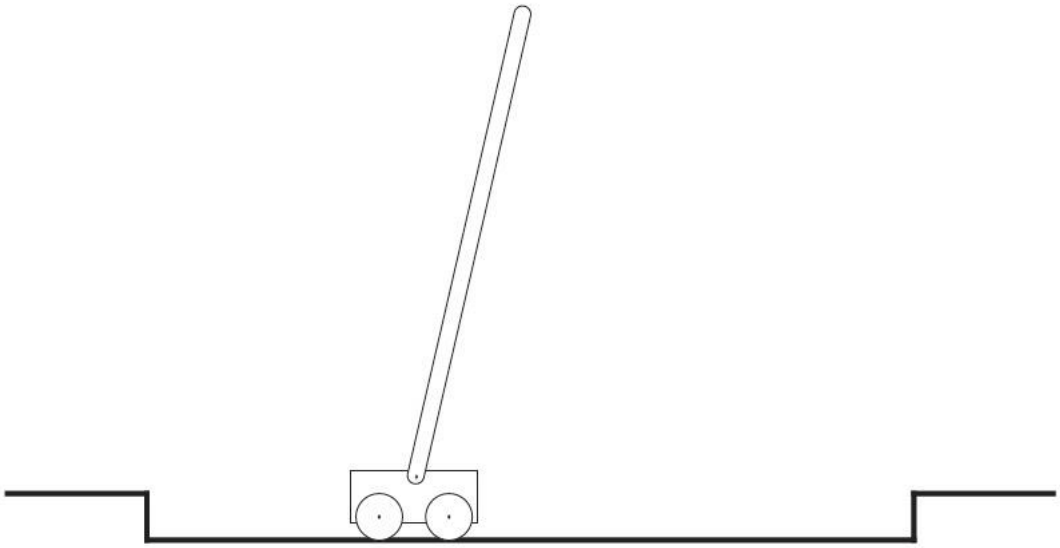
$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \qquad [1.0]$$

where T is a final time step. Such approach makes sense for applications where there is natural notion of the final time step, so that, when the interaction of the agent-environment naturally breaks into subsequence, called episodes, such as a play of the game, some sort of repeated interactions or trips through the maze. Every episode ends with a specific state which is called the terminal state, it is followed by the reset to starting state/to the sample from standard distribution of the starting states. Tasks with this kind of episodes are called episodic tasks. With such tasks we need to define the set of nonterminal states, which are denoted as S, the set of all the states and the terminal state, which is denoted S^+.

On the other hand, in many cases the interaction of agent-environment doesn't naturally break into episodes which can be identified but goes continually without any limit. For instance, this would be a natural way of formulating continual task of process-control, or the application to robot with long-life span. Such tasks are called continuing tasks. Return formulation is kind of problematic for continuing tasks since a final time step will be T = 1, and return, could be easily infinite. (For instance, assume an agent receives the reward of +1 for each time step.)

[32] Watkins, Christopher J.C.H. (1989). *Learning from Delayed Rewards*. King's College, Cambridge, UK

Figure 9. The pole-balancing task



source: Reinforcement Learning: Theory and Practice (Csaba Szepesvari). Accessed: 12 April 2020

The additional concept that we need is that of discounting. Aligning with this approach, an agent is trying to select the set of actions so the sum of discounted rewards it is going to receive in future is maximized. Especially, it chooses it 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. Discount rate[33] is needed to determine the present value of the 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 case each of the actions made by the agent happened influencing only an immediate reward, not a future reward, then the myopic agent may maximize it by maximizing separately each immediate reward. Generally, to act for maximizing immediate rewards may reduce the access to the future

[33] Hasselt, Hado van (2012). *"Reinforcement Learning in Continuous State and Action Spaces"*

rewards so that return may be reduced. As γ approaches 1, the future rewards are taken into account stronger: an agent is more farsighted.

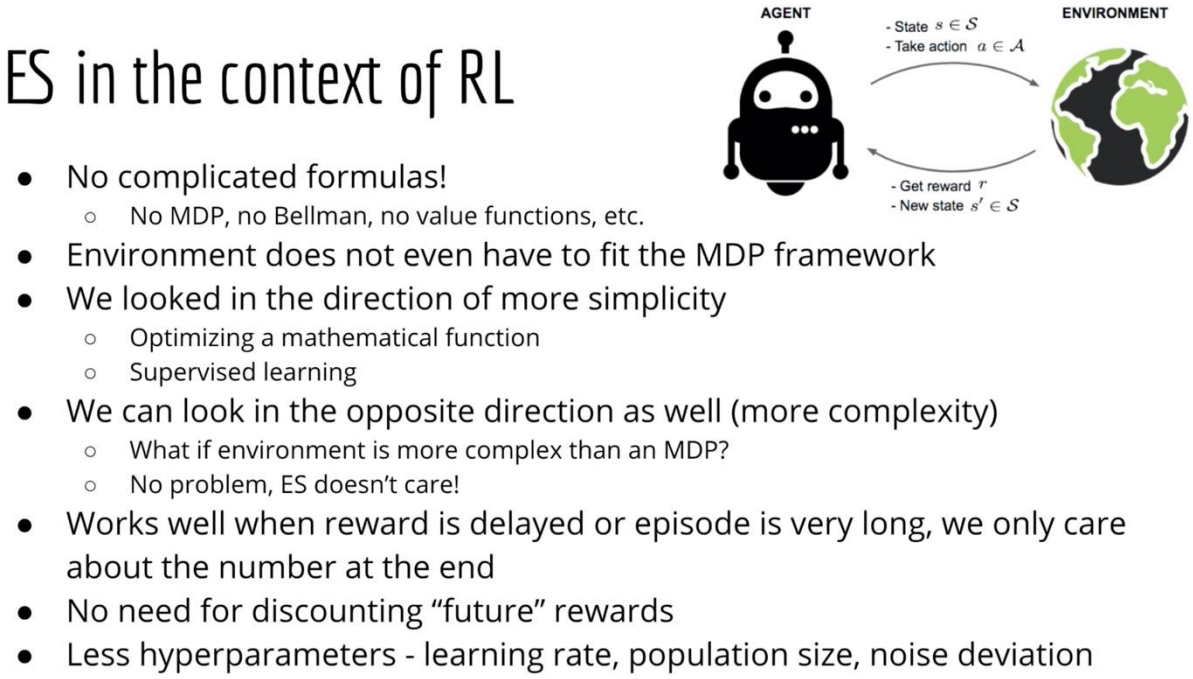
4. Implementation of agent playing "Flappy Bird" game

4.1 Evolution strategy:

For our experiment we are going to use evolution strategy[34] algorithm[35] 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 10. Evolution strategy in the context of Reinforcement Learning



source: udemy.com. Accessed: 12 April 2020

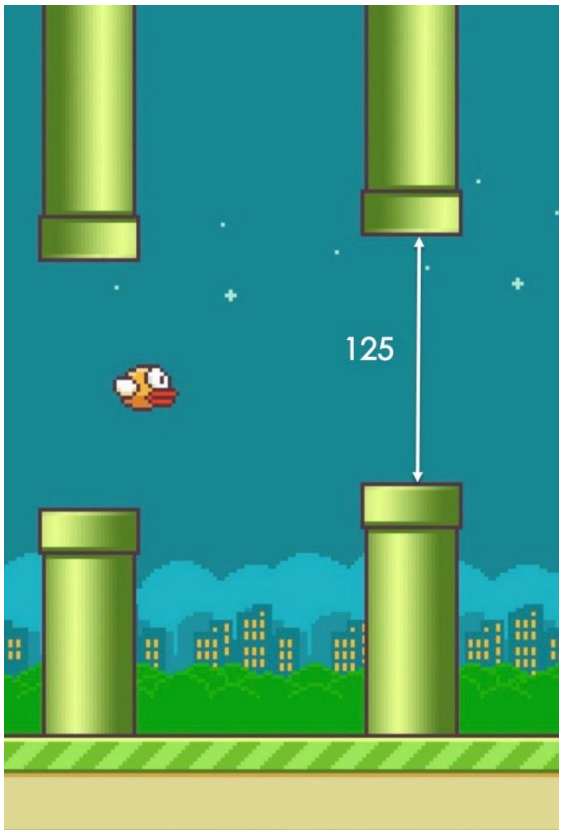
Our environment is a mobile game Flappy Bird where the player (or the agent in our case) has to maintain the position of the bird as long as it can. Every time the bird falls down – game starts from the beginning. The task may seem difficult for humans but not for the agent.

[34] Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Accessed: 12 April 2020

[35] Simon, D. (2013): Evolutionary Optimization Algorithms

This is how the game looks like:

Figure 11. 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. Initial distance between the pipes was chosen 125 pixels (shown on the screenshot above).

4.2 Evolution Strategy theory

Let's start with some motivation with biological evolution:

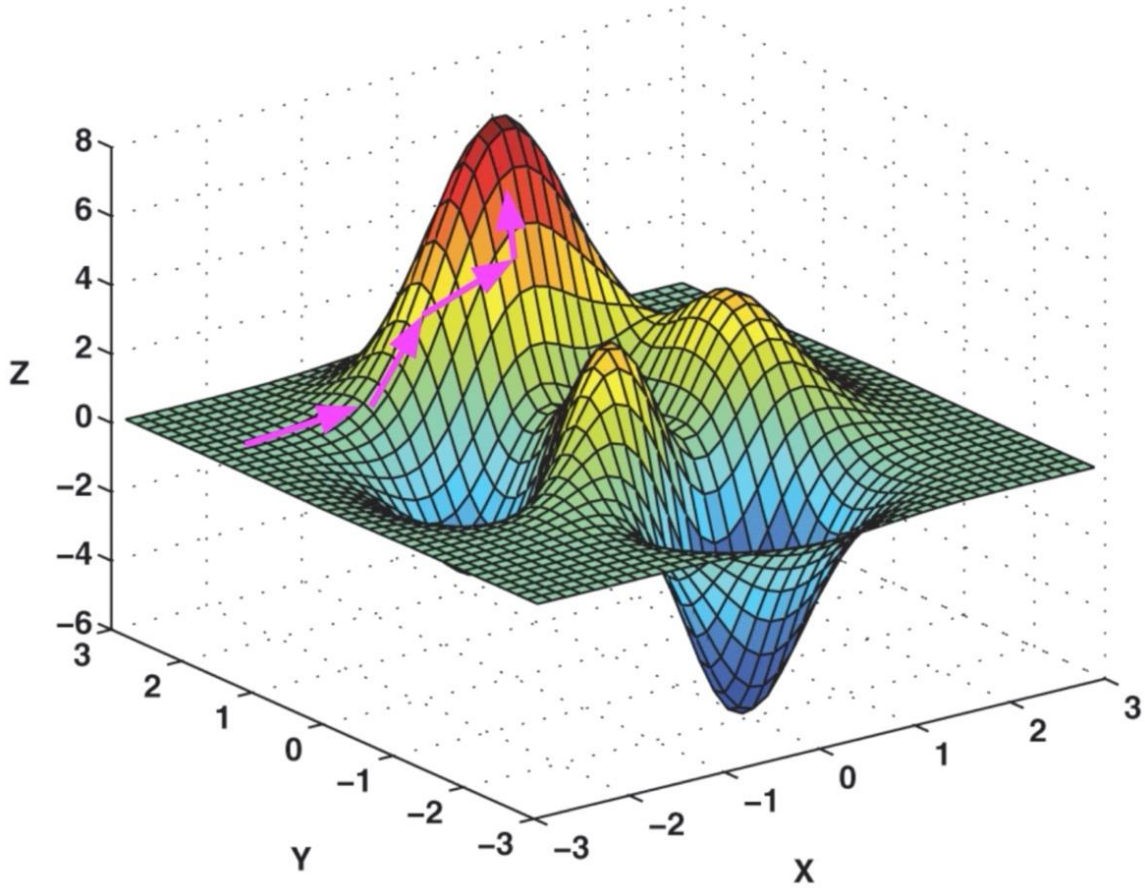
- DNA is a code that describes us (hair color, height, etc.)
- Assume we are dealing with single celled organisms that copy themselves to produce offspring (rather than 2 parent organisms which is more complicated)
- DNA isn't copying perfect!
- Mutations can be good or bad:

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 12. Illustration of algorithm



source: udemy.com. Accessed: 12 January 2020

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:

fig.13 – Illustration of algorithm

```
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

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

source: udemy.com. Accessed: 12 January 2020

First, we have to introduce a few more variables:

- Learning rate η (a hyper-parameter controlling how much the weights of the network are adjusted)
- 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.

4.3 Training parameters

Now, since we have a strategy for implementation, we just need to specify initial parameters.
For my experiment I've chosen population size = 50 (number of offsprings), learning rate = 0.03, sigma = 0.1 and number of iterations = 500, pipe gap = 125. 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.

5. Conclusion:

Model with such parameters was training for 1 hour and 54 minutes which proves the effectiveness of Evolution Strategy algorithm since it is significantly lower amount of time than was used for training agent with Q-learning: 6-7 hours. Running model with the trained agent afterwards showed that it was able to achieve amazing results, flying more than 3 minutes without falling down. 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.

6. Abstract

The goal of the thesis is to prove that Evolution Strategy algorithm may be used for solving Reinforcement Learning problems and be as sufficient as older, proven to be effective algorithms, such as: Q-learning, SARSA, DQN, DDPG, A3C. The main disadvantage of abovementioned algorithms is that they are highly relying on MDP[36] (Markov Decision Process) framework which is very computationally demanding due to recursion involved for solving its main equation - Bellman Equation[37] (also known as a Value Function). Thus, all these algorithms can't be used efficiently if the one does not have the access to a bunch of GPUs which are usually required in order to train a good agent. With my experiment I'm going to prove that even having 8GB RAM laptop may be enough for RL task using Evolution Strategy algorithm since it does not use Bellman Equation and, hence, not based on recursion which may require tremendous amounts of computational power.
The experiment will be conducted on the example of mobile game "Flappy Bird".
Implementation of the same game using Q-learning algorithm took about 6-7 hours in order to train the agent (the link to this experiment attached in "Literature" section). I claim that with Evolution Strategy a good agent may be trained in a shorter period.

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"

[36] Richard S. Sutton & Andrew G. Barto (2014). *Reinforcement Learning: An Introduction*, The MIT Press Cambridge, Massachusetts London, England, p. 67-69

[37] Bellman, R.E. (1957). *Dynamic Programming*. Dover. Accessed: 12 December 2019

7. Literature:

1. Richard S. Sutton & Andrew G. Barto (2014). *Reinforcement Learning: An Introduction*, The MIT Press Cambridge, Massachusetts London, England, p. 67-69. [Online]
   Available at:
   https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
   [Accessed 12 December 2019]
2. Bellman, R.E. (1957). *Dynamic Programming*. Dover. [Online]
   Available at: http://smo.sogang.ac.kr/doc/bellman.pdf
   [Accessed 12 December 2019]
3. "A Google DeepMind Algorithm Uses Deep Learning and More to Master the Game of Go | MIT Technology Review" (2016). *MIT Technology Review*. [Online]
   Available at: https://www.technologyreview.com/2016/01/27/109178/googles-ai-masters-the-game-of-go-a-decade-earlier-than-expected/
   [Accessed 12 December 2019]
4. McCartney**,** Scott (1999). *Eniac: The Triumphs and Tragedies of the World's First Computer***,** Walker & Co
5. Rosenblatt, Frank (1962), *Principles of Neurodynamics*. Washington, DC:Spartan Books. [Online]
   Available at: https://link.springer.com/book/10.1007/978-3-642-70911-1
   [Accessed 14 December 2019]
6. Arthur, Samuel (1959). *Some Studies in Machine Learning Using the Game of Checkers.* IBM Journal of Research and Development. 3, p.210–229 [Online]
   Available at:
   http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.2254&rep=rep1&type=pdf
   [Accessed 14 December 2019]
7. Altman, Naomi S. (1992). *An introduction to kernel and nearest-neighbor nonparametric regression.* The American Statistician, p.175–185. [Online]
   Available at:
   http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1010.2854&rep=rep1&type=pdf
   [Accessed 18 December 2019]
8. Elliot, David L. (1993), *A better activation function for artificial neural networks*
9. Barzilai, Jonathan; Borwein, Jonathan M. (1988). *Two-Point Step Size Gradient Methods*. IMA Journal of Numerical Analysis. P.141–148. [Online]
   Available at: http://pages.cs.wisc.edu/~swright/726/handouts/barzilai-borwein.pdf
   [Accessed 18 December 2019]
10. Martin Anthony (January 2001). *Discrete Mathematics of Neural Networks: Selected Topics*. SIAM. pp. 3
11. Haykin, Simon S. (1999). *Neural Networks: A Comprehensive Foundation*
12. Adomavicius, G.; Tuzhilin, A. (June 2005). *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions"*
13. Richard Szeliski (2010). *Computer Vision: Algorithms and Applications*
14. (2015). *"Feature Engineering: How to transform variables and create new ones?"*. Analytics Vidhya. [Online]
    Available at: https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/
    [Accessed 18 December 2019]

15. Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*. p.237–285. [Online] Available at: https://arxiv.org/pdf/cs/9605103.pdf [Accessed 22 December 2019]
16. Watkins, Christopher J.C.H. (1989). *Learning from Delayed Rewards*. King's College, Cambridge, UK.
17. Hinton, Geoffrey; Sejnowski, Terrence (1999). *Unsupervised Learning: Foundations of Neural Computation*. MIT Press.
18. Vapnik, V. N. *The Nature of Statistical Learning Theory*, Springer Verlag, 2000.
19. Kaplan, F. and Oudeyer, P. (2004). *Maximizing learning progress: an internal reward system for development*. Embodied artificial intelligence, p.629. [Online] Available at: http://www.pyoudeyer.com/ims.pdf [Accessed 28 December 2019]
20. Clarke, Frank H.; Loewen, Philip D. (1986). *"The Value Function in Optimal Control: Sensitivity, Controllability, and Time-Optimality"*. SIAM Journal on Control and Optimization. p.243–263
21. Hartmann, A.K. (2009). *Practical Guide to Computer Simulations*. World Scientific.
22. Joseph L. Doob (1990). *Stochastic processes*. Wiley. P.46-47. [Online] Available at: http://bactra.org/prob-notes/srl.pdf [Accessed 28 December 2019]
23. L. C. G. Rogers; David Williams (2000). *Diffusions, Markov Processes, and Martingales: Volume 1, Foundations*. Cambridge University Press. p.121–124. [Online] Available at: https://www.cambridge.org/core/books/diffusions-markov-processes-and-martingales/027C86D2818FE57AE55099ED94557EB2 [Accessed 4 January 2020]
24. Zhou, X. Y. (1990). "Maximum Principle, Dynamic Programming, and their Connection in Deterministic Control". *Journal of Optimization Theory and Applications*. p.363–373
25. Burnetas, Apostolos N.; Katehakis, Michael N. (1997), *"Optimal adaptive policies for Markov Decision Processes"*, Mathematics of Operations Research, p.222–255. [Online] Available at: https://www.researchgate.net/publication/238836142_Optimal_Adaptive_Policies_for_Markov_Decision_Processes [Accessed 4 January 2020]
26. Ashlock, D. (2006), *Evolutionary Computation for Modeling and Optimization*, Springer". [Online] Available at: http://index-of.co.uk/Artificial-Intelligence/Evolutionary%20Computation%20for%20Modeling%20and%20Optimization%20-%20Daniel%20Ashlock.pdf [Accessed 4 January 2020]
27. Elijah Polak (1997). *Optimization : Algorithms and Consistent Approximations*. Springer-Verlag". [Online] Available at: https://books.google.pl/books?id=uoXuBwAAQBAJ&pg=PA754&lpg=PA754&dq=27.+Elijah+Polak+(1997).+Optimization+:+Algorithms+and+Consistent+Approximations.+Springer-Verlag&source=bl&ots=9ZJCaybNvB&sig=ACfU3U3IGq0-nxMNq2mevWcyEb_E0TjYQA&hl=pl&sa=X&ved=2ahUKEwjt5smqnNzpAhUM0qYKHR2eBYEQ6AEwBHoECAsQAQ#v=onepage&q=27.%20Elijah%20Polak%20(1

997).%20Optimization%20%3A%20Algorithms%20and%20Consistent%20Approximations.%20Springer-Verlag&f=false
[Accessed 4 January 2020]
28. MacCrimmon, Kenneth R (1968). *"Descriptive and normative implications of the decision-theory postulates."* Palgrave Macmillan, London.
29. Brans, C. H. (2005). *"The Roots of scalar-tensor theory"*. [Online]
Available at: https://www.researchgate.net/publication/1969681_The_roots_of_scalar-tensor_theory_an_approximate_history
[Accessed 4 January 2020]
30. Berridge KC, Kringelbach ML (1 June 2013). *"Neuroscience of affect: brain mechanisms of pleasure and displeasure"*
31. "Robot density rises globally" (2018). Robotic Industries Association.
32. Watkins, Christopher J.C.H. (1989). *Learning from Delayed Rewards*. King's College, Cambridge, UK
33. Hasselt, Hado van (2012). *"Reinforcement Learning in Continuous State and Action Spaces"*
34. Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*". [Online]
Available at: https://link.springer.com/chapter/10.1007/3-540-61108-8_27
[Accessed 12 January 2020]
35. Simon, D. (2013): Evolutionary Optimization Algorithms
36. Implementation of "Flappy Bird" with Q-learning algorithm [Online].
Available at: https://sarvagyavaish.github.io/FlappyBirdRL/
[Accessed 12 April 2020]
37. Introduction to machine learning for beginners. [Online]
Available at: https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08
[Accessed 14 April 2020]

Appendix A

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


# when HL = 1 then that means that state is equal to a current observation
# when HL > 1 then we concatenate current observation with the previous one to obtain the state
HISTORY_LENGTH = 1

# Wraps PLE and Flappy Bird so that it behaves more like OpenAI Gym


class Env:
  def __init__(self):
    # initializing the instance of FlappyBird class
    self.game = FlappyBird(pipe_gap=125)
    # then pass this object into PLE constructor and create an instance of that
    self.env = PLE(self.game, fps=30, display_screen=False)
    # init does some necessary things under the hood
    self.env.init()
```

```
    self.env.getGameState = self.game.getGameState

    # by convention we want to use (0,1)
    # but the game uses (None, 119)
    # action takes a value of 119 to push the bird upwards
    # 119 is ASCII number for w which is used in games for upward/forward movement.
    self.action_map = self.env.getActionSet()  # [None, 119]

  # function which takes an action
  def step(self, action):
    action = self.action_map[action]
    reward = self.env.act(action)
    done = self.env.game_over()
    obs = self.get_observation()
    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
    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
# input size = dimensionality of the data
D = len(env.reset()) * HISTORY_LENGTH
M = 50  # hidden layer size
K = 2  # output size


def softmax(a):
  c = np.max(a, axis=1, keepdims=True)
  # subtracting the max of each column since we don't like exponentiating the large numbers
  e = np.exp(a - c)
  # then we divide it by the sum of the exponentiated values to give us the model output probabilities
  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

  # initialize neural network's weights
  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 action_sample(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 all parameters of a neural network as a 1D array
  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):
```

```
    # takes 1D array of parameters and shapes them back into neural network weights and then
assigns
    # them to the neural network
    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,  # noise standard deviation that gets added to the parameter for each offspring
        lr, # learning rate
        initial_params,
        num_iters):

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

  full_time = datetime.now()
  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))

  print("Full time training: ", (datetime.now() - full_time))

  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
  # env.set_display(True)
  model.init()
  params = model.get_params()
  best_params, rewards = evolution_strategy(
      f=reward_function,
      population_size=50,
      sigma=0.1,
      lr=0.03,
      initial_params=params,
      num_iters=300,
  )

  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))
```