Experimental Studies in Pursuit of Experiential Robot Learning

Author: Advisor:

Ahmed A. Aly **Joanne Bechta Dugan**

A Dissertation

Presented to

The Faculty of the School of Engineering and Applied Science

University of Virginia

In partial fulfillment

of the requirements for the Degree

Doctor of Philosophy

in the

Computer Engineering Program

December 2019

APPROVAL SHEET

This Dissertation is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Computer Engineering Program

Author Signature: Ahmed A. Aly

This Dissertation has been read and approved by the examining committee:

Advisor: Joanne B. Dugan, Ph. D.

Committee Chair: Zongli Lin, Ph. D.

Committee Member: Gabriel Robins, Ph. D.

Committee Member: Gianluca Guadagni, Ph. D.

Committee Member: Michael E. Gorman, Ph. D.

Accepted for the School of Engineering and Applied Science:

Craig H. Benson

School of Engineering and Applied Science

December 2019

December 2019

ABSTRACT

Computer Engineering Department

UNIVERSITY OF VIRGINIA

Doctor of Philosophy

Experimental Studies in Pursuit of Experiential Robot Learning

by Ahmed Aly

Robots are currently not mature enough to be used in unconstrained environments (i.e. in the wild) because they cannot learn and thus cannot respond to new situations. Our hypothesis therefore is that the development of a methodology that permits experiential learning could allow robots to learn and therefore to succeed in novel situations. We developed a method called Experiential Robot Learning (ERL) that outlines how robots should be developed. Neural Networks (NN) provide a promising path towards ERL and this dissertation evaluates this promise. Experimental studies illuminated a problem with using NN for ERL: the need for a differentiable loss functions and architectures can't always be satisfied, and the exploitative-nature of gradient-descent is not suitable to solve problems that require exploration. To address these shortcomings, we developed Local Search, a NN training approach that provides good results in the absence of a differentiable loss functions, or a loss function entirely, and on problems that require exploration. Our work paves the way for more advanced robot implementations adhering to ERL method.

ACKNOWLEDGEMENTS

I would like to thank Professor Joanne Bechta Dugan for her unwavering support and unyielding commitment to quality & academic integrity. My family has been my rock upon which I can lay all heavy troubles and receive concrete support to maintain balance. Friends helped along the way to make every day and every night bearable and passable. When things went wrong, one can always use a friend. When everything was great, providing support and comradeship was rewarding and wholesome. Finally, to the person without whom this thesis would not have existed, Rehab I am in love with you. Thank you, everyone.

This journey has been the hardest thing I ever did in my life. I am proud of it, and I apologize for all my shortcomings. I wish I made use of every resource I had available to me, and every helping hand. I did what I may, and today I am humbled by that experience. It has been shown to me, that the best way forward, is to believe in humanity. Our common humanity. I thank the NSF and the American education system, and the American government, and in turn the American people, my people, without which none of this could be accomplished. America invested in me as a person, without waiting for a reward.

I hope one day I can justify America's investment. To show that similar opportunities need to be handed out to all who seek them. God bless you and God bless these United States of America.

TABLE OF CONTENTS

CHAPTER 1: ROBOT LEARNING

INTRODUCTION

In today's world robots are markedly absent from everyday life. Robots are generally limited to industrial settings and such environments where there are strong restrictions on the degrees of freedom of both the robot and the environment. For example, consider a robot on an assembly line where it would be assigned one job on a conveyor belt, say to grab unidentified objects. That robot wouldn't be expected to perform tasks or jobs outside this particular one, at least without reprogramming. Other agents, e.g. humans, wouldn't interfere with the conveyor belt or the robot's sensory inputs. In addition, the type and size of "unidentified objects" is also restricted. The robot cannot expect to encounter an airplane for instance on its belt, or an explosive device. Thus, in this simple example, the robot would conditionally operate in a highly structured environment where there are strong restrictions on its degrees of freedom, as well as the environment's degrees of freedom. This condition can be traced back to the unique challenges of operating in unstructured environments, especially one as chaotic and complex as humans'. One cannot enforce such strong restrictions on the living environment of a typical human being, however.

We perceive a future where robots can become ubiquitous in commercial and residential settings. Self-driving cars are examples of such a future. The robot is the car acting in an unstructured and probably chaotic environment, making decisions constantly. Despite imposing strong restrictions on what drivers cannot and should not do, it is impossible to prevent unexpected and chaotic behaviors. There are many agents within the environment

outside of drivers, for instance pedestrians and the weather. The robot thus has to adapt to unforeseen situations and operate in this unstructured environment.

Our vision for the future features robots that will be able to not just co-exist in the real-world natural environment, but to manipulate it and themselves as well. A housekeeping robot should be able to grab dishes, fold laundry and open doors. All these actions involve an interaction with an unstructured environment. The shape of dishes, laundry and doorknobs is not predetermined. The robot needs to manipulate these objects, either with or without guidance from a human. Furthermore, the housekeeping robot must navigate indoor environments which can feature different flooring, e.g. tiles and carpets, and lighting. Object manipulation, Perception and Planning thus must be acquired in some capacity rather than exclusively pre-programmed. The Roomba robot (autonomous vacuum) is a primitive example of such robot. It is able to navigate the indoor environment using pre-programmed behaviors and clean the floor. These are not always the best behaviors. For example, bumping into objects and backing off is poor behavior if said object is expensive, fragile and prone to toppling. However, it can also learn the indoor floor maps in order to be able to cover the area effectively and wholly. The Roomba is primitive in both its scale of operation and its capacity to learn. A more advanced Roomba should be able to analyze the environment and adapt its behaviors accordingly. If a behavior is poor, e.g. it leads to a breaking of an object, it should be able to stop it and acquire a new behavior. In this simple example, the Roomba should realize which objects it can safely “bump into” and which it should “go around”.

There are several challenges to this vision. First and foremost is the ability to learn, or lack thereof. We define learning as the ability to improve one's performance on a given task. Without learning, robots will only be able to handle situations for which they have been programmed. If a self-driving car is programmed to self-drive only on highways, it will not be able to drive on a typical road. If a housekeeping robot is programmed only to fold laundry, it has no chance of opening doors. Furthermore, it will only be able to fold the sort of laundry it was programmed to. If it was programmed for towels, then bed sheets are out of question. Another challenge to our vision is the lack of supporting hardware. It is true that robots have come a long way, hardware-wise as well as the embedded systems within. However, there is still a lack of sensors as complex as a human's eye for example. The data streams, e.g. video from a camera, provided by typical sensors can be regarded as primitive. Thus, even if we had a learning robot, it may have a hard time learning from the given data due to its low quality, low (or absence) of information density.

Despite these challenges, robots can still accomplish much. Automated factories are one such example where robots can be invaluable to humans. Tesla, a US automaker, recently opened the Gigafactory where its Model 3 cars are produced. Originally, the Gigafactory was supposed to be entirely automated. There were many challenges to this scenario, Elon Musk, Tesla CEO, famously coined it as "*production hell*". The company had underestimated the difficulty of the task and overestimated the capabilities of robots. Things like tightening bolts, applying glue and fitting parts together are rather difficult for robots to perform accurately and quite easy for humans [1], [2]. Due to such challenges, the company resigned to including humans in its assembly line and reducing automation.

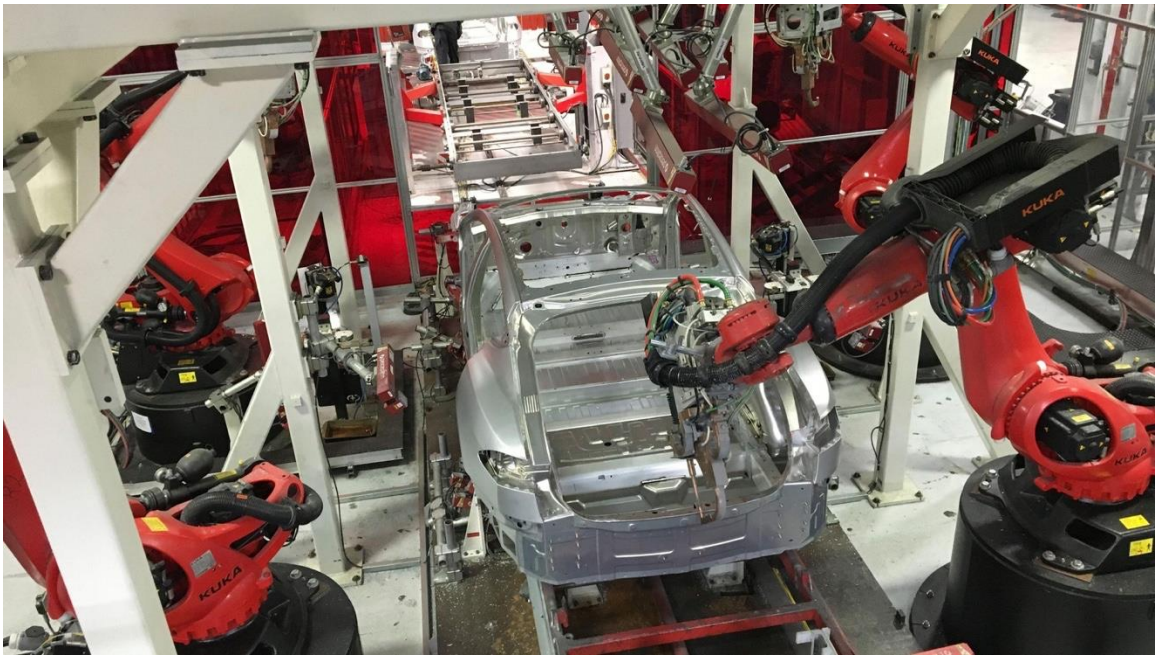


FIGURE 1 ROBOTS MAKING ROBOTS, A PHOTOGRAPH FROM TESLA GIGAFACTORY ASSEMBLY LINE [1]

Thus, even while not entirely independent, robots can still become a powerful aid to humans. In addition, robots can improve the quality of life for humans through alleviating hardship of special needs and disabilities. Housekeeping robots can help people who are mobility-challenged. Self-driving cars can help those who are vision-challenged. There are many such examples of the transformative nature and impact autonomous robots can have.

Our vision for the future is not just robots that will take actions, but robots that will *learn* from the actions it takes. Robots that will face a unique and new problem and will work "creatively" to find a solution. If a house robot drops a cup of water, the next time it holds water it should have figured a better grasp. If an outdoor robot slips on black ice, the next time it encounters black ice it should adjust its gait or move slower. These autonomous robots should mimic a human's ability to improve performance through experience, i.e. to learn.

MOTIVATION

It is important to note the reasons for focusing on Robot Learning as opposed to other sects of scholastic inquiry in the area of Robotics. Before the inquiry set forth in this thesis, our focus was on Human-Robot Interaction (HRI). This is a sprouting field that concerns itself with the design of interaction between humans and robots. Upon investing time in that domain, it was clear to us how meaningless the interactions were between humans and robots. Often the robot, though with physical form, had no capacity in the environment. It could not manipulate objects, nor operate in real-world open environments. Furthermore, without proper perception the robot (any robot) could not appreciably perceive the person with whom it interacts with as well as the manner of their interaction. For example, the robot could not yet shake the hand of a human in an unstructured setting. With advancements in Computer Vision, much of these challenges will be mediated. However, as it stood, the status quo of HRI revealed a certain lack in robot autonomy and capacity.

There can be two distinct approaches to address this limitation. The first is to improve Perception, Planning and Control algorithms. This has been the traditional approach. With each improvement, some aspect or problem is tackled and overcome. Incrementally, the robot equipped with such algorithms would build capacity and improve its ability to act independently. We call this approach hand-coding, hand-programming or manual-programming because solutions are developed to specific instances. The second approach is to equip the robot not with state-of-the-art algorithms but with some capacity to develop them. Robots that learn (or the computers underneath) would autonomously develop their

own Perception, Planning and Control capabilities. Consider the case of humans as an example. Children, generally, have a similar capacity to acquire visuomotor skills but end up developing differently based on many factors including personal experience. The second approach would resemble this but in robots instead of children, or in "child robots". We call it Experiential Learning.

Let us examine a case where such Experiential Learning would be a valuable inclusion. Annually, a robot soccer competition called RoboCup is organized. Teams of researchers from many universities and labs around the world participate in this prestigious competition with their team of NAO humanoid robots. The goal is simple. To win, the team must defeat other teams by co-operating to defend their goal and score goals of their own. In each iteration of RoboCup, the competition gets tougher. Playing conditions are adjusted to be more like those in real soccer games. Thus, organizers give the researchers more challenges to their teams. For example, in RoboCup 2016 the playing field flat carpet was substituted with a rugged carpet that mimics trimmed grass. The color-coded ball, which is easier to perceive, was substituted with a regular soccer ball. In such ways, researchers are forced to deal with these environmental changes.

Through my participation in RoboCup with a team from Taiwan, it was clear why hand-coding is a limited as an approach to realize robots that will perceive and interact with their environments. The difficulty posed by a rugged carpet as opposed to a flat one should not be understated. The mobility of the NAO robot was significantly hindered, and even blocked, as a result. The robot would trip and fall often when making sharp turns. The

research team had to re-tune and update parts of their control algorithms. Even more troublesome was the change from a color-coded ball to a normal black-and-white one. The Computer Vision algorithms had to be discarded because the new ball would many a time blend in with the white field lines, leading to the robot losing “sight” of the ball. Again, the team needed to develop new algorithms that would work with this new situation, a time-consuming process. Here the limitation of the hand-coding approach was apparent.

The mission statement of RoboCup is to advance state-of-the-art in Robot Perception, Planning and Control, and to one day face a team of the best human soccer players; and perhaps defeat them. The reasoning is simple, keep increasing the challenge of the environment until robots play in actual real-world conditions. The organizers expect to reach that level by 2050, in 30 years.

FIGURE 2 NAO HUMANOIDS PLAYING SOCCER AT ROBOCUP 2016, LEIPZIG, GERMANY. I TOOK THIS PHOTOGRAPH WHILE ATTENDING ONE OF THE SEMI-FINAL MATCHES. MY TEAM WAS ELIMINATED IN THE LEAGUE STAGES OF THE COMPETITION.

Other examples that defined our motivation to work on Robot Learning were the 2015 DARPA Challenge, and the Fukushima Nuclear accident. In the DARPA Challenge, humanoid robots were expected to accomplish several tasks in a structured, yet challenging, environment. Those tasks included getting out of a car, opening a door, climbing stairs and operating a valve. Needless to say, the robots failed in many ridiculous, and sometimes hilarious, ways. Though these robots were prepared by world-class labs, tested and programmed. None of those robots was learning, however. All were “set”, fixed, rigid, pre-programmed.

FIGURE 3 HONDA'S ASIMO ROBOT DANCING IN DISNEYLAND. [3]

In the aftermath of the Fukushima nuclear accident, some questioned why the popularized Honda robot ASIMO wouldn't contribute to cleaning the rubble or search-and-rescue efforts. The company responded that despite its appearance of "intelligence", ASIMO was not prepared to operate in that environment [4]. The robot would shake hands, approach people, and seem to comprehend simple conversation. However, it was limited by what it was programmed to do. In other words, by how "we" programmed it.

For all those reasons, in this inquiry we want to pursue a different approach. We want robots to learn autonomously from experience. We want robots to be able to self-improve so that "our programming" is not the explicit constraint to their learning, but the quality of the learning algorithms themselves. Whatever learning algorithm(s) we will come up with

it will also have limitations, it will not be able to learn everything. As such, we will need to constantly develop and improve those learning algorithms. Therefore, our reasoning in following the second approach is not to eliminate research, allow the robots to learn everything, raise our hands and dub our "job" complete. Our reasoning follows that of Deep Learning (DL), which we will outline in the following paragraph.

Deep Learning was popularized by the ability of a Deep Neural Network (DNN) to win the ImageNet image classification competition. The DNN outperformed all the competitors by a relatively strong margin. The approach of DL was simple. The features of the images were not "hand-coded" into the classifier. Rather, the classifier "learned" the image features automatically. In hindsight, we conclude that the amount of "feature engineering" that can be put into a classifier by humans does not compare at all to what the classifier can learn on its own through DL. Instead of engineering the classifier, DL researchers focused on engineering the learning algorithm and the neural network. To summarize, autonomous learning from the image dataset far surpassed what humans can put into the classifier.

Our reasoning behind adopting the second approach, i.e. Robot Learning, follows a similar path. We reason that what the robot can learn on its own will far exceed what we can "program" into it. Even if at the moment the best learning algorithm cannot match the performance of a mediocre "hand-engineered" algorithm on a specific task, in the long run this would not be the case. Again, we appeal to the case of Deep Learning vs. feature-engineered Computer Vision approaches.

Problem Statement

Problem Definition

Typically, robots have been designed following a task-oriented approach. The system fulfills a specific, usually narrow purpose. If any unforeseen changes occur to its operating environment or conditions, the system usually fails to achieve its goal.

In addition, following task-oriented approaches usually yields systems that are incapable of learning and adapting. The solutions do not extend to other domains. For example, the robotic assembly line in industry has been around for decades. These robots did not contribute to more advanced robotic solutions directly, as their implementations are application-specific.

Finally, when designing robots to operate in civilian environment that is incredibly complex and dynamic, hand-coded solutions can be a limiting factor. The limitation is latent within the environment. We illustrated the example of self-driving cars earlier. If those cars were incapable of learning and adapting to different circumstances, they would unquestionably fail to drive on roads. The reason is simple, the engineer cannot foresee every single possible scenario that the environment would inflict upon the system.

Summary: Hand-engineering approaches limit the generalizability and scalability of robotic solutions. We see it as a limiting factor to the prevalence of useful robots providing services in civilian life.

Elements and Scope

Considering the previous discourses on the nature of the research problem, we use this section to elaborately identify its elements and scope, culminating in a formal problem statement. It is important to identify what lies in the scope of this inquiry and what does not, due to the subtle and engulfing character of the problem.

We begin with an outline of the elements of the problem. The first element is the fact that a Robot is involved. The robot should take a physical form, yet there are also "virtual" robots that exist in simulation. In fact, there is no cutting clear definition of what a robot is and is not. We define a robot to be an embodied entity with ability to directly affect its immediate environment. Being "embodied" distinguishes it from a cloud computing intelligence, i.e. a robot has to exist in a bounded mobile form not as part of an abstract entity. The latter part of the definition distinguishes a robot from a computer. The latter cannot move and affect the environment directly, e.g. push an object, despite being contained. An excellent discourse with more detail on this subject can be found in [5]. Our definition also brings forth the notion of Agency. The robot has agency within its environment. It also has varying levels of autonomy. We shall not define the different levels of autonomy and how they are distinguished from one another. However, it will suffice to state that in our study, we are only concerned with autonomous learning. That is, the robot should be able to do the act of learning independently of human participation. However, this does not mean that we should not avail ourselves of human expertise. Indeed, the robot learner can still make use of human "instruction" or "participation" whilst

operating independently. If a robot asks for a humans' evaluation or judgement, it does not entail that the robot lost its autonomy.

Having thus defined what a robot is for our purposes, we turn our attention to learning. The robot must learn. We follow the same definition of Learning as in [6]. Learning is defined as an improvement of some performance measure P over a task T with experience E. Whatever constitutes "experience", "improvement" and "task" we don't attempt to define explicitly. Rather we use the intuitive and apparent meaning of those terms. It should be noted that in this context, however, learning is not the task. The task is not the learning process itself, but rather some external assignment/problem that the robot tries to do/solve. Thus, if we want the robot to walk then walking itself is the task, not learning how to walk.

With those two elements defined, we now presume to define their interaction. How does being a robot affect the process of learning? And how does learning affect the behavior of a robot? First, recall that a robot as we defined must feature embodiment. This embodiment allows a robot to gather data from interactions with its environment. The gathered data provides the learning algorithm with a constant supply of information that can possibly be utilized in the learning process. In other words, being embodied potentially allows the robot to be an active data "gatherer" for the learning algorithm.

Second, being a learning entity means that the robot should not abandon a task just because it is presently unsolvable. The robot should realize that it still can solve the task or improve its present solution. Whether the robot's persistence continues indefinitely or not should depend on the nature of the task and the purpose of the robot. Some tasks by nature are

transient and will force a limit on how much the robot will spend solving them. Other tasks are open-ended and may consume the robot's time/energy resources entirely.

This of course cannot continue indefinitely, thus a cap on how long the robot will persist is required. This is not a cap on learning, i.e. the robot should still learn. Rather it is a cap on how much the robot will spend attempting to solve a task or improve its solution.

Summary: Robots are embodied agents that can demonstrate improvement at performing a task by attempting different things over time, i.e. learn. Learning and Embodiment affect each other in different ways. Therefore, care should be taken when designing a system that facilitates robot learning.

Conclusion

We presented on how hand-engineering approaches to providing robotic solutions are unscalable and limit the prevalence of useful robots in civilian life. We discussed the core motivations of this thesis, and why robot learning should be the main line of focus. This helped give some perspective and define a scope around the problem.

A solution to the hand-engineering approach would be a method to develop robots that are capable of learning. This method should be focused on learning in the real world, and be platform-agnostic so that it's not for a particular type of robots. I hypothesize that such a method with these characteristics will contribute to eliminating the need for hand-coding approaches by unlocking robot learning. This method should be validated via experimental studies, which should generate new insights and pave a path for advanced implementations.

CHAPTER 2: EXPERIENTIAL ROBOT LEARNING METHOD

Following the discourse in the previous chapter there is a clear definition of what the problem is, as well as the characteristics of a potential solution. To address the limitations of hand-coding approaches, an alternative method should be proposed. This method would outline how to develop robots, according to what principles and using which techniques.

Through a series of observations and deductions, we propose a method inspired by human-learning for creating developmental robots. This method, which we call **Experiential Robot Learning** (ERL), has two component an Approach and an Implementation. We draw reference from the literature in [7]–[9]. The ERL method addresses the issue of robot learning on two levels, a high level and a lower level (of abstraction). The ERL method outlines goals which we aspire our robotic solutions to resemble, such as being Scalable.

The Approach component addresses the higher level. It outlines the principles according to which robotic solutions should be developed. These principles create a general frame of reference. If need be, perhaps due to the available technology, one may deviate from them. The ERL Approach should be adhered to as closely as possible, regardless, in order to produce the desired effect of addressing the limitations of hand-coding approaches.

The Implementation component addresses the lower level. It prescribes the use of Deep Neural Networks and how they can be implemented. Neural Networks (NNs) of themselves don't have bearing on ERL. If there is an alternative that retains the desirable characteristics of NNs, there is no prohibition to use that instead. We, however, have not found such an alternative. The experimental studies that will follow evaluate the use of NNs for ERL.

ERL Approach

Instead of articulating a rather lengthy and possibly confusing discussion, we elected to define the ERL Approach through a series of bullet-points. The bullet-point format is chosen to allow the reader to survey the different items with speed. Under each bullet-point a brief discussion is presented so as to capture its essence concisely. We do not assign a significance to the order of appearance of these bullet-points. Again, the items listed below constitute principles to which robot developers should adhere as closely as possible. There are no hard requirements on what must be followed, and what can be good-to-have. The scope of the problem we are addressing prohibits imposing such restrictions.

Principles

1. ERL is Open-ended Learning

Open-ended learning means we are not targeting a specific skill for the robot to learn. Nor is there a problem the robot can solve. Furthermore, it means there is no beginning or end to learning. It is a continuous and gradual process. We aim from this that the robot can maximize its potential to learn through experiences. It means, as well, that a robot immersed in a certain environment can acquire different skills than an identical twin, immersed in a different environment.

2. ERL is horizontally and vertically Scalable

Scalability is a key component to our approach as we believe it is a fundamental limitation in hand-engineering approaches such as those used in the RoboCup competition. The

solutions yielded by task-oriented paradigms are usually too-domain specific. That is, solving one problem does not lead to solving another. On the other hand, if the approach is scalable then it could potentially solve a myriad of problems.

To tie this back to robotics, consider the following. If a walking algorithm can only function for smooth terrain, it means that another algorithm needs to be found to traverse rough terrain. In contrast, if there is a general walking algorithm, like the one human beings possess (in general), then it can solve traversing smooth, rough and any other terrain traversable by humans. This is what we attempt, finding a general rather than a domain-specific solution. If our endeavor is successful, we do not solve the problems of Robotic Vision singly, or Robotic Motion singly. We would have solved much more than single problems.

We identify two dimensions to scalability:

a. *Horizontally Scalable*

   By this we mean that the same approach is applicable across-domains. We define domains here as learning different modalities such as Vision, Speech and Motion. Ideally, we want the same approach to be applicable when learning natural language as when learning to perceive different objects.

   In addition, we also want the method to be applicable across problem/application-spaces. That is, to use the same approach when the robot is learning to navigate a human indoor environment as when it is learning to grasp objects. This saves the engineering effort of trying to solve every single problem separately.

b. Vertically Scalable

The robot/agent needs to be able to improve with time its functions/skills. That is, if at first it learns how to walk a wobbly walk, it must learn to walk better and better with further experience of Walking. Ideally, it can then learn to walk on uneven surfaces and learn to regain balance if tripped or pushed.

Again, there are prime examples of this problem being solved very well using a hand-engineered method such as Google's Atlas [10]. That system did not learn how to walk or perform functions autonomously. It has been designed to do many tasks such as carrying objects, and designed to do them very robustly. However, again, due to this approach, it cannot learn new concepts or skills. It is an ingenious application of bleeding-edge control theory.

3. ERL is Platform Agnostic

The developed solutions should be applicable to robots of different shapes and capabilities. In this way, we are not again just hand-engineering a solution for a specific platform.

4. ERL is Hierarchical/Accumulative

The learning is accumulative in a similar manner to a child. The child must first learn about obvious concepts as discussed earlier, and then through those be able to learn more complicated concepts and abstractions. The method also defines that learning can be hierarchical in the sense that the more abstract concepts may require fusion of learnings across modalities. Learning to grasp an object requires first learning to recognize objects and learning the grasping motion, separately.

Being hierarchical/accumulative should not be confounded with a hard requirement on accumulation of learnings in a linear manner. Learning a new task may possibly degrade performance on an older task. The robot may need to re-learn the older task. Being hierarchical/accumulative means that in order to unlock the potential learning of "more advanced" tasks, the robot may first need to learn its constituents.

5. ERL is Physically-Experiential

Learning is stimulated and induced by physical experiences. That is, the robot/agent must be immersed in the physical environment and learn through this immersion. Some experiences will be more stimulating than others. For example, a robot's perception would benefit from exposure to different environments rather than being confined to a single room. In this context, interaction with a human, or other elements in the environment, is also an experience; possibly the most important one. The range of experiences will always be subjective as some experiences the robot/agent may not be able to learn from. For example, a visually-perceptive robot will not benefit from auditory experiences.

LIMITATIONS

We also anticipate the following limitations to developing learning robots. These items should be pondered and regarded before and during the development process. They directly affect the type of problems that can be solved with ERL, and the manner in which they're solved (through experiential learning).

1. Computational Resources limit ERL

The computational requirements to perform the function of learning may surpass the capacity of current embedded systems. It means that the computations may need to be executed remotely on a server. This is an emergent approach called Cloud Robotics. Otherwise, the scale of the learning system would need to be reduced. These considerations also include the battery capacity and power requirements of the robotic solution.

2. Offline Learning limits ERL, but may sometimes be necessary

Online learning means that the learning process is happening as the experience is unfolding. For example, when a child attempts to touch a hot object, the learning is happening as the hand approaches the object and after it, i.e. throughout the experience.

Offline learning means that the learning happens after the experience has finished. An example would be if a robot explores a space, then returns to a human-coach and attempts to label objects it did not recognize. The learning of the new objects in this sense occurs after the robot had encountered them. To clarify: humans, for example, learn online during their waking hours and offline during sleep. The type of "learning", i.e. the expected result of learning, depends largely on whether it occurs online or offline. A human may learn online to play Tennis and adjust their grip. Yet it may be that only during sleeping does the mind reflect on the events of the day to generate insights; the events "sink in" as one may colloquially say.

Robots that learn from experience should do most of their learning On-Line/On-site/On-the-job. This is learning-while-doing. The amount of Offline learning, as described above, should be eliminated because it reduces the ability of the robot to learn throughout the

experience. It may be necessary though. Some tasks, as we've presented, may be unsolvable without offline learning.

3. Lack of Responsiveness limits ERL

This subtle limitation pertains to online learning. We use responsiveness according to the application at hand. For example, we interact with our mobile devices in real-time. Even though all the events are discrete and take calculable time to occur, we perceive a swipe on the screen as a generally responsive experience. It responded quickly to the human gesture.

A robot learning online, can still be unresponsive. For example, due to computational demands, learning may take minutes to perform the necessary computations. This would be perceived by a human as an off-line learning process, even if it is not. If the goal involved conversing with the robot, such lack of responsiveness would render this goal impractical. Another case would be if the robot was required to interact with non-stationary objects. If the robot's learning systems can't respond quickly enough, the task (and consequently the potential for the robot to learn) would fail.

Therefore, in such a way, the ability to respond timely must be considered as a limitation to what the robot can learn. Whether the robot's learning process occurs "responsively" or not is what we mean here, depending on the application.

4. Supervision limits Scalability of ERL, but may sometimes be necessary

The presence of a teacher, a guide or a coach for the robot is a major obstacle to its self-propelling potential. Effectively, the robot becomes bound by what the human can/will

teach it rather than being an independent entity. The robot also has to be capable of receiving these teachings from the human, and use them effectively. To counter this, there are approaches of unsupervised and semi-supervised learning. Those topics of research are less mature than supervised techniques [11].

The notion of robot schooling has been introduced before [7]. This notion assumes that robots with little to no learning will require more supervision from human teachers. As the robot matures and gains knowledge about the environment, the amount of supervision needed decreases. This appeals to the aforementioned characteristic of ERL being hierarchical. The robot would maintain a generally-increasing trajectory of self-sufficiency.

5. Unpredictability limits ERL and makes it less robust

Another limitation to the ERL Approach is the unpredictability of the outcome. It is unknown what the robot can or can't learn. It is unknown how long learning would take. It is unknown what the interaction will be when attempting to fuse different modalities or skills. It is unknown how much, if at all, learning at the early stages of the robot's development affects learning afterwards. Being experimental by definition, we don't provide theoretical guarantees on convergence.

Additional Limitations

From the use of neural networks, and training techniques such as Deep Learning, arises four additional limitations to ERL, as outlined below.

6. ERL is limited by the lack of Knowledge Transfer

It is unclear whether learnings can be transferred or not. In one way, it may be possible, perhaps by importing the weights and network structure of the network that has learned certain skills in different environments.

In another way, it may be impractical because the imported network has adapted to the particular agent/environment.

This argument is similar to the topic of Sim2Real, where a network (controlling an agent) first experiences a simulated environment, then it is imported to a different situation (arguably a physical agent in a physical environment). There is typically a "reality gap" that would need to be bridged. To be clear, we're not advocating Sim2Real. It's only mentioned here in attempt to illustrate an example of how Knowledge Transfer may work.

If importing the network's weights/architecture is successful cross-domain and cross-environment then this may potentially save a lot of time/effort. We need only one agent to know how to walk successfully, for example, and then it would mean that all our robots can walk. Else, all our robots would need to learn how to walk individually. In this way, ERL is limited by the lack of Knowledge-Transfer.

7. ERL can be limited by the available Data

Deep Learning and similar techniques are known to be bound by the amount of and entropy in the data. If data is limited, learning also becomes limited. In a sense, data resembles experiences, or rather data encodes experiences. Entropy in the data refer to what the data

encodes in relation to the representation capacity of the neural network. If the representation capacity is high and the data itself is quite correlated, then there is great potential for overfitting. The network simply memorizes sequences and patterns rather than learning the features needed to generalize to new experiences. This of course depends on whether Data is needed at all for learning how to solve the task

8. ERL can be limited by insufficient Pre-training

As mentioned earlier, the network may need to be pre-trained so as to cognitively bootstrap the agent. The amount of pre-training, depending on the task, may be insufficient for the robot to learn from experience. For example, in a face detection task, if the network is not sufficiently pre-trained, then the robot would not be able to find a solution. In such a way, pre-training or lack thereof may become a limitation to ERL.

9. ERL is limited by the absence of Reasoning

We do not believe nor claim that training neural networks with state-of-the-art techniques is currently able to deliver any measure of reasoning. Therefore, we do not target cognitive reasoning as one of the goals/skills for our robots to achieve/learn. In addition, to counter the effect of lack of reasoning, we can build state-machines to transport the robot through the different phases of learning. The robot/agent decides the transitions autonomously, according to the pre-defined behavioral rules.

This constitutes a significant obstacle to learning from experience, because the robot would arguably be able to learn much more effectively should it be able to “reason”. Engineering

clever systems/state-machines instead to anticipate possible outcomes and direct the robot accordingly is also a limitation to ERL scalability.

ERL Implementation

In the previous section we have defined the ERL Approach through a series of bullet-points of principles and limitations. What remains is to define how it would be implemented. The ERL method leverages Deep Neural Networks to realize the learning behavior. DNNs have attractive properties that are conducive to the ERL approach. The robot becomes a cognitive agent, an embodiment or physical manifestation, by which the neural network improves. The improvement takes place by autonomously having the network modify its architecture and/or weights in response to learning signals.

For example, if a Convolutional Neural Network (CNN) learns a new class of objects, it expands the last Fully Connected (FC) layer and retrains itself to recognize members of the new class. Such is the autonomous learning behavior we are targeting; and believe can be realized with neural nets. This flexibility is indispensable to our method. In a way, it is not unlike the brain reinforcing/creating connections between neurons to solidify new learnings.

The use of Neural Networks is not particular to ERL in the sense that should another learning structure prove itself more attractive, we can readily use it instead. That alternative structure needs to be conducive to the Approach of ERL at least as well as NNs. The key characteristics of NNs are its malleability and tractability.

It is theoretically proven that neural networks are universal function approximators [12]. This means that they can be used to compute any function whatsoever. Neural Networks are also simple enough in structure to be simulated by today's hardware with up to 10^9 parameters. This computational tractability is crucial to performing experimental studies.

Neural Networks Introduction

The concept of computational neural networks was devised many decades ago. However, from the 2011 ImageNet competition submission by Alex Krizhevsky, commonly called AlexNet, they received newfound attention [13]. Perhaps one of the earliest and most interesting works in the domain was that of Yann LeCun on handwritten digit recognition [14]. He presented an architecture of a type of Neural Networks called Convolutional Neural Networks (CNNs), or ConvNets for short. It is based on the operation of convolving input images with "filters".

Each filter is typically a few pixels in size, and is comprised of random distributions. As the filter is convolved with the image, patterns either emerge or are suppressed. This depends on the shape/distribution of the filter and the image. In this way, the first few layers produce abstract 'maps' of curves and edges. Those maps are then combined to form different shapes, resembling the first steps to recognizing entities. Eventually, the distinguishing features of the entity in question start emerging after a few convolutional layers.

Figure 4 provides an example of those maps. Most prominently notice the map on the upper right corner. It clearly is activating with the body shape of the horse. The map on the upper

left corner is reacting to the background foliage. Other maps seem illegible. Note also that there is a map that is entirely black. This means that the image of the horse does not correspond with whatever feature it is responsible to detect, if anything at all. Some maps/neurons/nodes in the ConvNet are redundant.

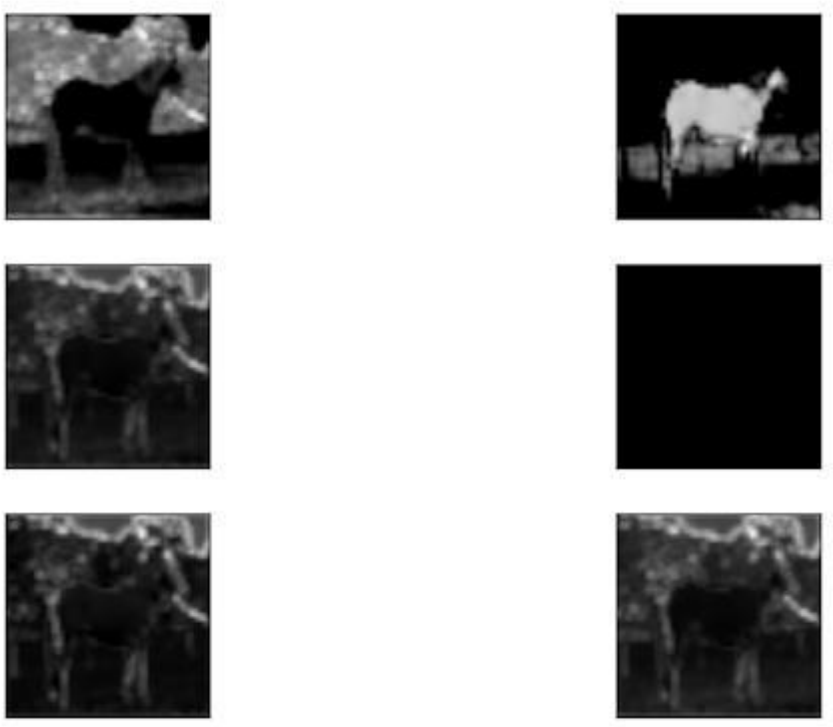


FIGURE 4 ACTIVATION MAPS' SAMPLE FROM A HORSE-RECOGNITION CNN. THE MAPS SHOW HOW NODES/NEURONS FIRE UPON DETECTING DIFFERENT FEATURES SUCH AS THE HORSE'S BODY ON THE UPPER RIGHT.

Neural Network Training

Training a network means adjusting the weights of the connections between the nodes/neurons to enable a function. The function of the network is implicitly encoded in what is called a loss function. It thus becomes an optimization problem in an extremely high-dimensional space to find global minima.

When training a neural network, one can start with a random set of weights or import weights from a pre-trained network. The values of those weights lead the network to produce an output when given an input. In a supervised learning context, the output is then judged either correct or incorrect. In an unsupervised learning context, it is simply accepted as is. However, a loss function is still computed for unsupervised paradigms such as Auto-encoders. The loss function there, is not a label on the *correctness* of the output. It is the distance between the produced output and the input. The criteria by which the network is trained, i.e. the loss function, is different for each type of neural net. Yet, the goal remains the same, i.e. optimizing the weights (and biases).

Data is commonly split between training and validation sets, and less commonly there is also a test set. The training set is what the network is exposed to during training. An epoch is one single pass over all the data in the training set.

For the solution to generalize, the validation set is used to evaluate the model. After each epoch, the loss is calculated for both the validation set and the training set. Both values reflect different meanings. Training loss is what is used to train the network. Validation loss reflects how well the solution generalizes to instances beyond the training data. The fear is that overfitting might occur, given the usually substantial entropic capacity of modern neural networks. It would manifest with the network's performance being exceedingly well for the training set, while stagnating for the validation set. Stagnation occurs when the accuracy stops increasing, or similarly when the loss stops decreasing, after each epoch. Overfitting usually implies that the network failed to learn the core

features of the data. It could have different meanings, though, such as indicating a problem with the data itself. The test set may be used as an extra step to evaluate the generalization capability of the trained network. The test set is generally exposed to the trained model only once after training is complete. It can act as a final assessment of generalizability.

BACKPROPAGATION

Training is traditionally done by error backpropagation. The basic algorithm for this is called Stochastic Gradient Descent (SGD) [15]. By computing the gradient, the direction in which the weights are adjusted is known. Recall that the solution space is extremely high-dimensional. The solver is minimizing the loss function, computed after each epoch. Loss is the distance between the current output and the desired output. SGD is thus used to determine in which direction the solver moves. This is not the only technique for training, but we discuss it here since it is the most prevalent.

There are more complex algorithms than SGD, such as ADADELTA [16] and ADAM [17]. They incorporate concepts such as Momentum and Learning Rate adjustment. In short, there are more features to reaching the solution than merely computing the gradients.

Note that the output is a nonlinear function computed over the entire network. That is, output is the nonlinear sum of the weights (and biases) in the network with respect to the input. Nonlinearity is essential, otherwise the solution space collapses. Nonlinearity is introduced using Activation Functions. At each layer in the network, an activation function on the output is used, i.e. multiplied. Thus, the neural network constitutes an affine transformation between the input and the output.

Rectifier Linear Units (ReLUs) are commonly used in modern CNNs. They have been shown to work best as the depth of the network increased. Weights that are less than zero are set to zero and positive weights maintain their value. There are explanations for this. Also, there are other concepts pertaining neural nets such as Vanishing and Exploding Gradients [6]. For the sake of succinctness and focus, however, this will conclude the exposition on the details of neural networks.

Neural Architecture

There are many types of neural networks. We believe we can leverage different types for different kinds of learning. For example, for learning to recognize objects usually CNNs are used. To learn motion sequences, we believe Recurrent Neural Networks (RNNs) can be used. In this way, to learn each modality engineering a certain solution is necessary.

Though seemingly similar, it is different from the hand-engineering approach of learning features. In our method, we engineer the solution. In the hand-coding approach, the problem is engineered to extract features which could then be used to produce a solution. The affordance of techniques such as Deep Learning is that features are autonomously learned [6].

The network, if pre-trained, would initially act as a cognitive bootstrap for the agent. This means that it needs to reflect some level of knowledge of the world before it is ready to experience it. Otherwise, the solution space will likely be too wide for any convergence on a skill or knowledge. In addition, the cognitive bootstrap acts as evidence that the network is indeed capable of learning the desired modality.

Modern neural architectures typically feature a large number of parameters, which can go up to 10^9 parameters. This creates a valuable opportunity for the robot to learn given the immense representational capacity. It is also alerting because such networks take longer to train, and constitute a large search space for optimization algorithms.

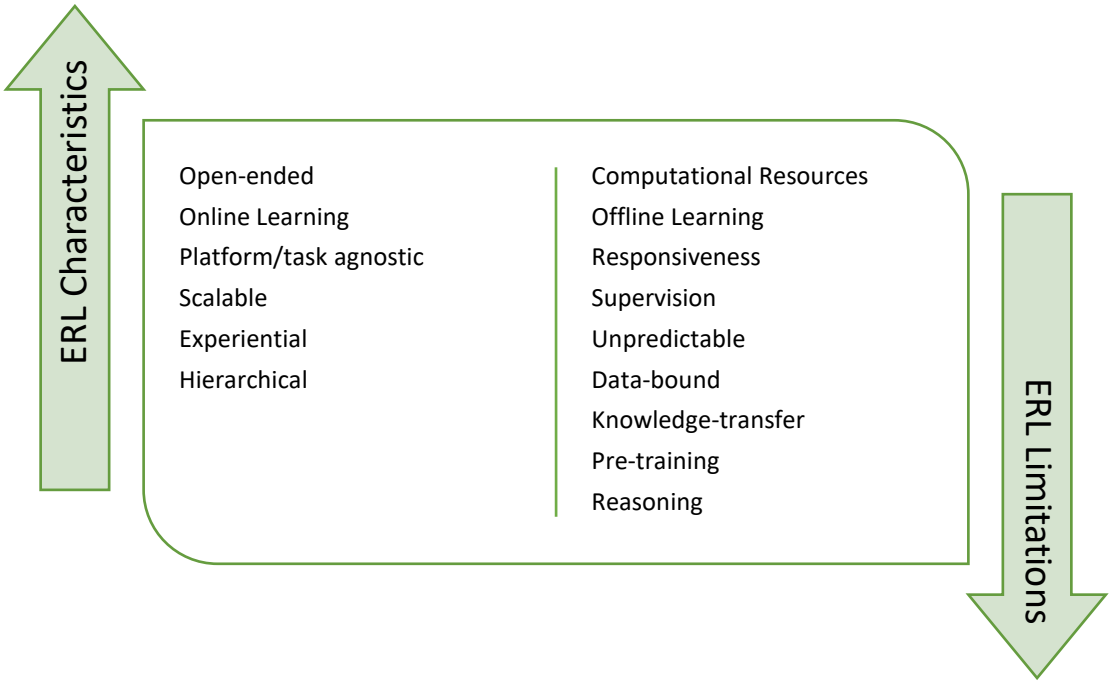


FIGURE 5 CHARACTERISTICS AND LIMITATIONS OF ERL

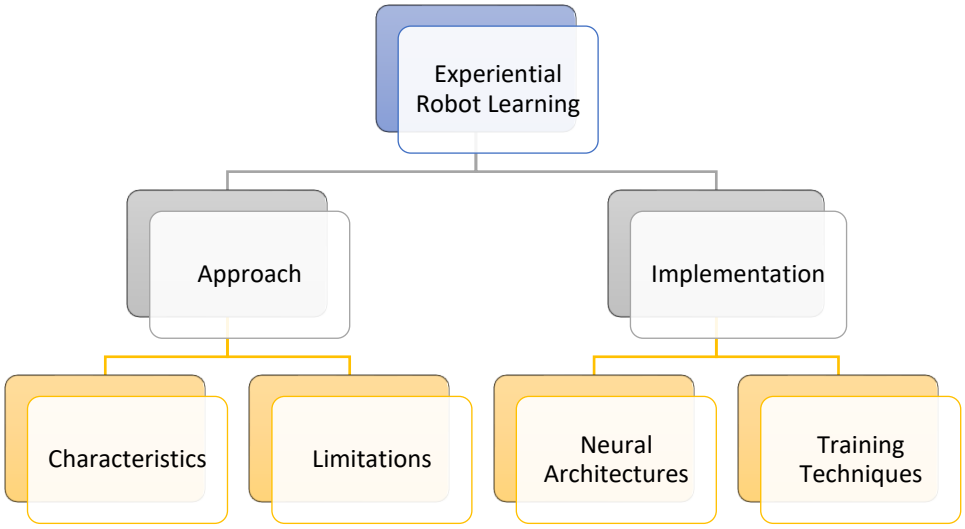


FIGURE 6 EXPERIENTIAL ROBOT LEARNING IS COMPOSED OF ITS APPROACH AND IMPLEMENTATION

Summary: We propose Experiential Robot Learning to create developmental robots leveraging Deep Neural Networks. The method is designed to work around the shortcomings of hand-design approaches. ERL consists of an **Approach component**, to set the design principles and define a vision, and an **Implementation component**, to use modern DNN architectures and training techniques. We presented the 5 Characteristics and 9 Limitations that define the ERL Approach, as summarized in Figure 5. An overview of ERL is depicted in Figure 6.

Thesis statement: Neural Networks trained according to the Experiential Robot Learning (ERL) methodology provide a viable path to the solution of robot rigidity resulting from hand-coding approach.

SUCCESS CRITERIA

We are undertaking existential, investigative cross-domain research. The primary concern is whether our propositions lead to learning or not. That is, do the experiences improve the robots' knowledge or not. How we define learning/improvement is specific to each experiment. Due to the outlined nature of our work, we constructed a proxy metric to assess the attractiveness of implementations following our method (ERL). We call it the Attractiveness Score (AS). There are multiple criteria against which to evaluate the attractiveness of robots developed following our method, outlined below. We combine these criteria in the AS to give a measure of how well the implementation generally meets the principles of ERL. Next to each criterion is an AS 'weight', assigned according to its importance.

1) Demonstrating the capacity to learn or acquire a new skill (AS: 5)

   This is the overarching measure of success. Therefore, it has the most weight in the Attractiveness Score. The robot needs to demonstrate some capacity of learning experientially.

TABLE 1 ATTRACTIVENESS SCORE FOR LEARNING

Achievement	Mark
Robot demonstrates learning capacity	5
Robot fails to demonstrate learning capacity	0

2) Time taken to achieve a new learning (AS: 2)

   It takes significant time for Reinforcement Learning approaches to converge on a policy[18]. In simulation, it means little and less because the program can run

indefinitely and arbitrarily fast. However, in a real-world environment, it would be folly to expect an agent to undergo an experience or try an action tens or hundreds of times before learning. Therefore, an important measure of attractiveness is how fast an agent can learn.

TABLE 2 ATTRACTIVENESS SCORE FOR CONVERGENCE

Achievement	Mark
Robot learns in first trial	2
Robot learns in 2-10 trials	1
Robot learns in more than 10 trials	0

3) Time taken to pre-train/bootstrap the agent (AS: 1)

From our previous discussion, we know that Neural Networks require pre-training to perform functions such as classification. If it takes a day or more to train a network so that it can support the agent, this will eventually become a constraint as the network expands and the agent becomes more intelligent.

TABLE 3 ATTRACTIVENESS SCORE FOR PRE-TRAINING

Achievement	Mark
Pre-training takes less than 24 hours	1
Pre-training takes more than 24 hours	0

4) Effort taken to gather data/bootstrap the agent (AS:1)

If the datasets required to train the neural networks must be built from scratch then the time taken to do so, and effort, will become a limitation. For example, consider if the network needed 30 minutes of motion-logging data. That would not become as large of a hindrance as if it needed 30 hours' worth of data. There are tasks for

which there are datasets available online, most notably Image Recognition. In contrast, learning a concept as Object Reachability, as we conceive it, would require compiling a labelled dataset.

It is not always clear how to measure the time required to compile a dataset. For example, consider a dataset of online cultivated images. One may get those images through an API in seconds, or get them through manual download one at a time. Each method has its advantages, and one may be required for the task at hand. It would be difficult, however, to accurately measure the time it took to download and curate those images by hand. For most practical purposes, we will assume that if the task of compiling a dataset spanned more than a day, then it took more than 24 hours regardless of the actual time consumed.

TABLE 4 ATTRACTIVENESS SCORE FOR DATASET COMPILATION

Achievement	Mark
Dataset takes less than 24 hours to compile	1
Dataset takes more than 24 hours to compile	0

5) Practicality of model being deployed on an embedded system (AS:1)

We are well-aware that deploying neural networks is computationally demanding on even the most advanced embedded systems. Practicality of the implementation refers to how plausible it would be to deploy the systems fully on the embedded boards, without a backend. This is a measure of attractiveness because the goal, ultimately, is to have local intelligence, without relying on external resources.

The constraints imposed by embedded systems on software sometimes require the designer to rethink the design. This measure gives an incentive to the designer to optimize his/her design to run as efficiently as possible.

TABLE 5 ATTRACTIVENESS SCORE FOR DEPLOYABILITY

Achievement	Mark
Model deployable on embedded platform	1
Model not deployable on embedded platform	0

In addition, since our work is concerned with *developmental* robotics, it is conceivable that a measure of lifetime learning is included in the attractiveness score. It would address the concern of whether learning stagnates after a period. However, the nature of that concern presumes there is an already-functioning system. That is not true, of course, least of all for this work. We therefore do not include such measure.

Success thusly is defined as obtaining an attractiveness score of more than 5. The weights have been designed so that this is not possible unless the robot achieves learning. That is, the sum of all the criteria besides learning is 5.

Summary: We present a proxy-metric to capture how ‘good’ an implementation following our method is, it is called Attractiveness Score (AS). The implementation must score more than 5 points to be considered a success. To achieve a passing score, the agent must demonstrate learning through experience. Thus, the AS scale captures the essence of ERL of learning from physical experiences.

Literature Review

We turn our attention to what others have done with regards to the problem, as stated in the earlier sections. By consulting the literature, it is apparent that so much attention has been devoted in the past few years to neural networks as a vehicle for learning. The term "**Deep Learning**" describes the field/technique of applying deep neural networks to machine learning tasks, mostly in a supervised manner using gradient-based optimization algorithms and error backpropagation. The attractiveness of neural networks arises from its ability to learn features in data automatically. This is also called **representation learning**, i.e. finding representative features in data. The representation capacity of neural networks refers to the volume of features that the network is capable of learning. A network with a higher representation capacity can learn more features, and vice versa. Thus, data with complex features requires networks with a sufficiently high representation capacity for learning to occur.

There are other implementations of learning besides neural networks such as Support Vector Machines (SVMs), Random Trees and linear regressors. Each of those implementations has its advantages and disadvantages. However, none match the representation capacity of neural networks. We know this from the field of **Computer Vision** (CV), where many techniques (e.g. SVMs) were used before neural networks. However, when the potential of neural networks was tapped, it became apparent that all were subpar in comparison. Thus, nowadays in CV neural networks are the primary research focus. For those reasons, we shall only focus on neural network implementations

of learning. There is a vast literature on robot learning, and learning in general, preceding the use of modern neural networks. It would be a daunting task, well beyond the scope of this section, to cover it all.

This section will hopefully clearly convey the novelty of our approach and how it compares to similar or related works. The section is divided into two subsections each conveying a different aspect about the research.

We define 2 dimensions of uniqueness to our method. Some works will overlap with our Implementation but not the Approach, and vice versa. No other work features a method or proposition as Experiential Robot Learning, to the best of our knowledge. A presentation on what the dimensions of uniqueness are, is given below.

Dimensions of Uniqueness

Approach

1) Motivation and Purpose

   We spent a large amount of effort outlining the vision, problem and intention for our method. Our motivations guide and shape the solutions we build. In this sense, other works that have followed the same recipe but had different motivations would more than likely produce solutions quite different to ours. ERL is rooted in the motivations and purpose we outline and cannot exist without them.

2) Scope and Context

We defined a broad scope for ERL. We don't intend to apply it to have robots learn a skill or modality, grasping for instance. This is a critical dimension. The scope for which the method is proposed deliberately puts it at a certain high level of abstraction. In addition, we specify the context within which we perceive ERL can be applied and the expected outcomes. For example, we target developing methods that can be applied to different robot morphologies i.e. being platform-agnostic. Again, this is an important dimension of uniqueness. Other works may target certain contexts/scopes such as human-centric robots would likely produce solutions that are distinct from ours at a high-level. It is more challenging to develop general solutions than specific ones. Being platform-specific one can neglect challenges typically faced by other platforms. Large degrees-of-freedom, for instance, are an issue for anthropomorphic robots but not quad-copters. I borrow a quote from my Mentor and professor at UVA, "*Everything becomes 'obvious' after it's done*".

IMPLEMENTATION

1) Modern Training Techniques

It appears that modern training techniques such as Deep Learning and Neuroevolution could facilitate ERL in a way that earlier approaches could not. The training dynamics and advanced practices, such as fine-tuning, specific to those techniques are crucial. Without those advanced practices, learning would not be online or autonomous. The training procedure can be supervised or un/semi-supervised; depending on the application. How to define modernity is not trivial.

Use of Neural Networks have been in study since decades ago [16]. An indicator of modernity is the ability to handle training of large networks with 10^6 parameters. Deep Learning and other modern techniques facilitate this. Thus, they allow for networks with larger representation capacities to be trained to solve more complex and significant tasks. We build upon and contribute to modern training techniques. Advances in those areas directly affect our implementations. That is, if there was a recent development (by us or others) that may improve the learning behavior of the robot, it may unlock a branch of learning we did not consider.

2) Custom Architectures

This is the most obvious element of uniqueness but perhaps the one we assign the least significance to. It is the way we use Deep Neural Networks. For example, in the first experimental study, we follow a certain procedure and implemented a CNN architecture based on VGG16 in a unique way. This adds to the distinctness of our work, which already stemmed from the other elements defined above. That is perhaps why it's the least important; it is an extension of our way of thinking and solving the problem, the final 'brick'.

We give all credit (through citations) and acknowledge, with respect, the work of researchers who tackled similar problems, and every source of inspiration to us. We did not conceive Experiential Robot Learning without other work to build upon and draw inspiration from. For instance, all the modern practices in Deep Learning (e.g. Fine-Tuning) were already published research before we thought about leveraging them.

Similarly, the literary work in developmental robotics abound. However, we hope that through the following discussions it becomes apparent how our propositions are distinct, novel and potentially of great impact and value to the research community.

Related Approaches

This section talks about the general approach of developmental robotics, particularly online, cumulative learning. We present work of comparable motivations and scope to ours.

First is the position paper in [5], which paved our way for thinking about the integration of robotics and DNNs. The paper clearly discusses the importance of using Deep Learning techniques to implement developmental robotics. It begins by discussing the importance of learning representations without hand-coding. After a brief introduction to different DL approaches, it swiftly goes over how DL can overcome challenges of building developmental robots. Finally, it touches upon the concept of learning through experience and artificial intelligent curiosity. As expected from a position paper, there is no implementation or experiment proposed, nor is a method outlined. We didn't find such items from the authors in later publications as well.

As a tangent to that paper, let us consider Curiosity, because can be an important enabler of autonomy in learning. Our work is just not yet ripe to incorporate high-level constructs such as curiosity into it. It would contribute enormously, however, to the autonomy of the agent. Especially, when combined with unsupervised Deep Learning techniques. If there was a successful implementation of Curiosity, it would perfectly fit within our vision for robotics.

These textbooks and papers in [7], [19]–[21] elaborate on the notion of intrinsic motivations for a robot. This overlaps with curiosity, or artificial curiosity as it is commonly referred to. Roughly speaking, it is about developing internal conflicting mechanisms of self-interest, i.e. motivation, and attention. These notions stem from developmental psychology literature [22].

We see curiosity differently. We see it as a *behavior* (in action space) implemented within the method we propose. For example, if the robot becomes proficient at scene parsing, it can detect and describe objects for which it has no label. The curiosity *behavior* would follow: The robot could autonomously build a new class label, i.e. unsupervised learning, and refine it. Alternatively, the robot could ask a human for the label and train itself to recognize the new object by walking in the environment and viewing it from different angles. This behavior can be implemented in a Finite State Machine manner, rather than using intrinsic motivations as proposed by other authors.

We leave the tangent of curiosity to return to work that followed similar approaches to ours. The outline of our method is generally mentioned in developmental robotics textbooks but without the scope and implementation. The nature of a textbook, however, meant that no practical implementation is presented; nor is a methodology advocated for any single purpose.

In textbook [8], there are discussions about creating *human*, not humanoid, robots with culture, law and religion. It is not our purpose since we only want to enhance robots so that they can aid humans, but it is related. In the course of [8], a discussion on how to bring

about such robots ensues. The authors detail the importance of learning and the use of neural networks. However, given the context of the book, propositions are broad and lack any practical implementation methodology or detail. Highly-abstract experimental results are given in simulation, such as navigating open environments to search for ‘food’, i.e. rewards.

In textbook [7], a narrative is given in the first chapter on the principles of developmental robots. Their characteristics, especially online and autonomous learning are outlined. To quote, they say “*Thus a truly online, cross-modal, cumulative, open-ended developmental robotics model remains a fundamental challenge to the field*”. The use of rudimentary neural networks, but not deep learning, is also introduced as ideas for implementations but no actual work is presented.

The previous two discussions were from relatively recent literature (2014 and 2015, respectively). To illustrate how dated these concepts are, we turn our attention to [23]. This textbook dating to 1987 discusses the principles of intelligent robots. It outlines the then-latest approaches in solving Perceptual and Reasoning problems. Without specifically referring to developmental robotics, the author introduces the notion of cognitive models and architectures.

In summary, autonomous, online learning is not an entirely novel concept in the domain of developmental robotics. The ‘infant’ field is rich with the works and proposals of many notable researchers. Namely, Cangelosi, Parisi and Di Nouvo. It has been rather consumed, however, with putting forth different cognitive models and architectures instead of

focusing on implementations. That is, most of the work in this sense is engaged in the formalism of cognition and cognitive development rather than producing cognitive robots.

Though incredibly important, we do not aim to formalize cognition. Neither do we propose cognitive models or architectures. The method we propose is more practical. Whether we are trying to 'run' before we 'walk' is a matter of question left to the reader. To develop ERL, we reason from the bottom-up, i.e. build implementations before proposing models, rather than from the top-down.

Our work borrows from several fields and sources inspiration. We present a clear methodology that relies on cutting-edge Deep Learning and/or other training techniques. There is a clearly defined scope, motivation and detail on applicability and implementation to guide the researcher/roboticist. We have a distinct target of building open-ended robots capable of autonomous learning through experience.

Paradigms Of Learning

We now refer our attention to the nature of learning for robots. There are different sorts of learning according to the manner in which learning occurs, and also the objective of learning. For a detailed discourse on this matter, refer to [9]. We shall present on some interesting learning paradigms, and how they relate to our course of study.

Lifelong Learning (LL) entails that the robot is able to learn throughout its "life", and that the learnings will accumulate without Catastrophic Forgetting. That is, learning one task (i.e. learning how to solve it) should not be extinguished after learning another. The learning must be accumulative. This approach though assumes that first the robot is able to

learn. With the capacity of learning secured, the robot does not learn to perform just one task, but rather an unbounded number of tasks. Thus, in Lifelong Learning, it is required that the robot is not only proficient at one task, but at several tasks at the same time. The tasks may or may not be related in structure, there is no clearly defined boundary on task similarity.

Transfer Learning concerns itself with the agent's ability to learn one task and use that knowledge to aid it in solving another task T2, where T1 and T2 are related. The robot should attempt to solve one task (or set thereof) at any given time. There is no requirement that solving one task in the robot's "lifetime" will help in solving another at a later stage.

Multi-task learning (MTL) refers to the type of learning where the robot's performance is not evaluated over one task T1, but over several tasks as a collective such that T = {T1, T2, T3, …, Tn}. If the collective of the smaller tasks is regarded as one bigger task, however, then the paradigm is not different from regular learning. The distinction of MTL is made to outline the usefulness of learning several joint tasks at once. It can lead to a better generalizability from the learning agent by preventing over-fitting to solving a specific task. MTL is different from LL in the sense that MTL does not incorporate the aspect of learning new tasks outside the pre-determined set.

Online Learning (OL) is an important and under-investigated paradigm of learning, especially as pertains this study. OL refers to the ability and act of learning a task on-the-fly, on-the-job, in-action, i.e. learning while doing. This is different from offline learning where data is first gathered, then learning ensues. Being "online" means that with every

incoming piece of data (or "experience") the agent updates itself somehow in attempt to improve. We don't adhere to OL strictly as a requirement of ERL but consider it as an aspiration when technologies and research are more mature and permitting.

The ability to learn on-the-fly, especially for robots, is invaluable because of the nature of robot interactions with the environment. Robots have the ability to act in an environment and affect it. The tasks given to robots typically are solved by affecting the environment in some respect. The process of robot learning thus is to find out increasingly better ways to accomplish those assigned tasks. With online learning, the robot can immediately leverage its current experience of the environment dynamics to come up with better solutions. This is not always possible. Furthermore, it is not always a "better option" than offline learning. For example, it can be inefficient to re-train an agent entirely every time there is new data. Thus, the online learning paradigm needs to be pondered with some reserve.

Let us discuss how these interesting paradigms of learning relate to our course of study. While they all are related with learning, not all are relevant to our motivation as outlined earlier. First, Lifelong Learning seems like an unreasonably high bar for robots to achieve at this stage. With the core piece, i.e. learning, still posing an incredible major challenge for robotics, it seems unreasonable to ask for more with current state-of-the-art. For example, how would one assess whether Catastrophic Forgetting has occurred or not? Would the robot be required to re-do all its previously-solved tasks in order to verify it is still able to do them proficiently? This is definitely an interesting paradigm to study, but we deem it impractical at this point. To work alongside humans in increasingly complex

scenarios, LL may be necessary. From our perspective, it is unrealistic to study this paradigm (especially experimentally) when the robot thus far can't even learn from experiences. Thus, we are not concerned with LL (and consequently Catastrophic Forgetting), but rather with the "ability" to learn autonomously.

Regarding Transfer Learning, it is not clear why learning one task would actually aid in learning a second task if the learning method, i.e. underlying algorithm, is the same. TL also assumes some similarity in the structure of the tasks, but there is no one universal way to evaluate task similarity. One can possibly evaluate this on several dimensions/metrics. Furthermore, in our motivation we underline robots capable of learning multiple tasks, but we did not assume that learning one task would actually lead to a faster learning of another. If that was the case it is certainly welcome, but we don't make it our primary arc of study. For those reasons, TL is also outside the scope of investigation.

Multi-task Learning poses an interesting paradigm, but it follows a similar line of reasoning to TL. That learning a group of tasks simultaneously is beneficial. It should be noted, that *learning* a group of tasks at the same time does not mean that they are performed in unison, i.e. simultaneously. That is, Multi-task Learning is <u>not</u> Multi-tasking. To learn a group of tasks, the robot performs each task separately then the performance measures are aggregated. The robot's overall performance is assessed over this aggregated measure. In our motivation, we don't outline any examples of robots performing multiple tasks (and learning from them). This may even be unfeasible to practically deploy on a real robot due to the amount of resources (time, energy) it would consumer. Imagine if learning a single

task was difficult, how much more difficult would it be to learn a group of tasks simultaneously? It seems unreasonable since we are still struggling with a single task. This line will not be pursued in this study.

Finally, and most importantly, is Online Learning. Unlike the other paradigms, OL is primarily concerned with learning one task which is in line with our stated motivation. OL however poses a strict limitation on the way of learning, it has to be online. While we would want this to happen, ideally, it may be infeasible or impractical. Learning may be more efficient if done in batches. Indeed, this is the typical scenario in most Deep Learning applications. Again, while desirable, we can't adhere to OL as a principle in ERL at this stage, but consider it as an aspiration.

To this point, we have addressed some learning paradigms and how they relate to this study. From this exposition, we outline that what really matters to this study is the problem of learning itself. Learning to do just one (but undefined) task, in any manner of learning.

In this thesis we aim to engage in the challenge of developing a robot that is capable of learning autonomously from experiences in its environment, to solve a single tractable task at any given moment. The learning can benefit from human expertise or instruction. The learning can be online or offline. We are not concerned with Catastrophic Forgetting. We engage in this research as means to address the aforementioned limitations in generalization and scalability imposed by the hand-engineering of robotic solutions. This stems from our viewpoint that the prevalence of competent developmental robots can provide useful services in civilian life.

Related Implementations

In this section, we present research that is comparable to ours in implementation but not in the approach. There are many papers discussing the use of neural networks in different domains of robotic learning and skill acquisition [24]–[26]. However, they don't present an overarching approach or methodology to extend their work to other domains. The scope of the work is limited, with varying degrees, to the application/subject in question. In developmental robotics, the application is usually broader in scope than other robotics contexts. Common applications are symbol grounding [27], sensorimotor fusion [28] and learning numbers [29]. This contrasts the usual robotics research paradigm of focused on Perception [30], Planning [31] and Control [32] separately.

First, we address the work in [33]. Their purpose is like the experiment we performed, i.e. open-set object recognition. They make use of RGB-D sensors to create a 3D point cloud plot of a scene. The data is then parsed through a feature-extracting procedure. They don't provide an accuracy metric which we can compare our performance against. In addition, the system is open-loop. That is, the robot is not "corrected" by a human if it makes a mistake. Neural networks are not used. The approach is task- specific, namely to recognize objects.

There are many works which do not use neural networks. They are in various domains such as Computer Vision [34], Learning numbers [35], Speech Recognition [36], Robotic Motion [37] and Knowledge Transfer [38]. Though these works are recent they do not

make use of either neural networks or deep learning. It shows that the richness in literature where a single problem can be tackled in many ways.

Similarly, there are many papers utilizing Deep Learning for various tasks. Those include Object Recognition [39], Object Detection [40], Generating Speech [41], Robotic Motion [42] and Navigation [43]. Most notable, however, is research using deep learning in the field of developmental robotics [44]. These researchers tackle a host of problems such as learning body representations [45] and equipping robots with the 'sense' of agency. The methods are mostly ad-hoc, since the field is yet in its infancy.

We turn our attention to some works of notable importance and relevance. The work by Di Nuovo [46] tackles the acquisition and manipulation of symbols and numbers. In [47] Luo addresses the problem of determining which direction a robotic arm moves in, and generates motion sequences for the arm to move in a defined direction. The robot thus learns the concept of direction in some capacity.

In the field of Reinforcement Learning (RL), Li [48] presents an approach to create open-ended intelligent robots. The authors advocate combining actor-critic models and reward-policy mapping. Their results are for a rudimentary navigation task, and only in simulation. The work in [49] also uses RL to have a physical humanoid acquire skills such as standing, leaning and walking. The humanoid, though, needs to experience a virtual environment in simulation until it acquires a reasonable policy. Meola [50] discuss using RL to generate different motions in robots, inspired by motor development in children.

The authors in [51] use two robots to classify 12 objects. Machine Learning Classifiers acquire features for object they classify. The paper stipulates that transferring feature representations is feasible and accelerates learning for other robots. Therefore, if one robot learns those features, other robots can use them and learn even faster. The two robots have different cameras and hardware, but they are the same class (mobile vehicle). This paper shows that learned low-level features are transferable across robots. This is not unlike what we already know about ConvNets. The low-level features, e.g. edges, acquired from learning one object help the network learn other objects faster.

In this paper [50], the authors discuss two kinds of motion, Rhythmic and Discrete. The former is where oscillatory neural signals, such as walking or waving, are generated and sent to the peripherals. The latter is where the signals are discretized/unique from origin to goal such as grabbing an object or pressing a button. During a child's early stages of development rhythmic movements are practiced first. An explanation for this is that rhythmic motion is controlled by an open-loop mechanism, with almost no feedback. The child would repeat the same motions as in a reinforcement learning policy. The paper proposes a model to explore the interplay between these different movements and carries an experiment involving a robot to verify it. The robot's human-like hand is used, with only the thumb and index fingers allowed to move. They implemented an efficient Reinforcement Learning algorithm and a reward function to help the robot learn to perform simple tasks such as rotating a dial. The paper is an excellent example of how child development can affect and inspire research in the field of developmental robotics.

This paper [52] presents an approach for learning the reward function from human demonstration. The demonstrations are sub-optimal and made by non-experts, which distinguishes this paper from others. Two simple experiments are presented. First a surgical robot called DaVinci is used to pick-up an object from an origin to a destination while avoiding an obstacle. The robot performs this successfully. It learns by computing an average path over the training dataset (compiled from 7 demonstrations). This becomes the policy for the robot (reward function). Paths are then generated based on this. The error is computed, which is the distance-to/deviation-from the learned policy. Therefore, without prior knowledge of the environment, the robot can pick up the object and reach the destination while maneuvering around an obstacle on the path. The second experiment is a sweeping task. A robotic arm (like the Baxter robot) is required to sweep a Green cube into a dustpan while avoiding two obstacles on the path. Again, from several demonstrations a training dataset is built. Paths are generated based on this reward function encoding the desired behavior, and an error signal is calculated for validation. It is shown that this approach can extend a step further. The robot is presented with a different setting of the environment (object, obstacles and goal locations), and it successfully maneuvers the cube around the obstacles to the goal dustpan. This paper is an excellent exposition on the learning from demonstration concept. Knowledge in this context is not acquired through direct, task-specific, rule-based approaches. It is acquired through experiencing the environment. The robot had learned that its policy was to avoid obstacles while reaching its goal destination.

On the topic of sensorimotor fusion, this paper [53] presents sensory fusion as a mathematical function of the sensory inputs. The concept is used to mimic human-balancing by a mechatronic robot. The control model is based on the Double Inverted Pendulum (DIP) model. There are two categories of signals, namely vestibular and proprioceptive. These signals are algebraically manipulated (fused) to produce composite signals/estimates. These estimates are used in the control loop to produce a balancing action on a tilting horizontal plane. Basically, the experiment consisted of gathering data from human subjects balancing on the tilting plane (4° variation) and contrasting it to that of a robot performing the same function. The robot and humans exhibit similar responses to the variation in the plane. This is taken as evidence that sensory fusion captures an essential component of human biomechanics. The robot features mechatronic limb systems (artificial muscles connected with springs) that allow it to be controlled by a model that is anthropocentric. We are shown here how sensorimotor fusion can capture some essential qualities in how humans perform physical skills such as walking or balancing.

Finally, in the Perception domain, this paper [54] presents the authors' work on porting a Convolutional Neural Network (CNN) to the Raspberry Pi 2 for Face Recognition. Training is done on a desktop computer with Nvidia GPU. Afterwards, the network is transferred to the Pi for Classification. As a benchmark, OpenCV algorithms were used to compare against the presented work. It is mentioned that the network outperforms all OpenCV algorithms (of which only Fisherbank was chosen for direct comparison) in terms of accuracy. In addition, the performance is similar at worst, and much better on average.

It shows that it is not impossible to deploy Deep Learning on embedded systems. This will be even more plausible in the next coming years as those systems develop and mature.

Throughout this section, we have attempted to present a survey of the works which relate to ours. The research overlaps many fields and areas hence the wide spectrum of papers. Specifically, however, we presented the papers under two distinct sections. The first was the Approach section where the authors adopted or proposed a similar approach or methodology to ours. Second was the Implementation section where the papers featured implementations like ours in some respect.

Broader Impacts

We leave the domain of fact and roam in the wilderness of speculation. The robots we create will be living, breathing entities in our world, if the ERL method is largely successful. Living and breathing in the sense that they could maneuver skillfully, they could see well and interact elaborately without being capable of reasoning. If the research proves fruitful and the method scales well, the robots can have the appearance of intelligence. They would be able to:

- Perceive the world, including us, as well as themselves
- Expand their perceptual knowledge
- Move about and navigate the environment
- Understand human speech and converse, without comprehension

We are likely thinking from an anthropo-centric point of view. These robots may as well develop skills beyond our expectations. This is outlined as one of ERL limitations. As rudimentary as those skills are, developing them can affect society in unexpected ways. It, most notably, can spur research in this domain, potentially leading to increasingly sophisticated robots. Even without comprehension, understanding or reasoning, non-experts will associate intelligence with such robots.

From the notion of the uncanny valley [55], we know that humans can and do perceive robots as a threat. Even in mere likeness, humans would rather not have artificial agents resemble them. A robot that moves about skillfully, perceives the world and, most importantly, learns and adapts will without a doubt be perceived as an introduction to the

Singularity [56]. It is a concept that there is a point after which AI will be self-sufficient, self-propelling without relying on humans. Cognitive Robots are the embodiment of AI. To most people, AI is mostly software, like Siri on their phones or Alexa in their homes. However, a robot that even seems intelligent, adds a whole new dimension to the mix. Being physically capable and able to interact socially, will have repercussions on society.

In addition, it has been well-known that AI can and will replace many jobs, resulting in a definite and broad impact on the US and global economy [57]. Human-dexterity has always constituted a barrier to deploying AI on agents, i.e. robots [58]. Many tasks require human-level motion [59], thus those tasks are currently only performed by humans.

An intelligent agent capable of operating in a real-world environment skillfully can replace human workers and farmers. There are jobs such as Power-line Maintenance which even affect the health of those performing it [60]. Consider also, for sake of argument, harvesting crops such as Blueberries. If robots can recognize the fruit, pick it and proceed to the next bush, it would mean that the task is not exclusive to humans (assuming this is presently the case). It creates a new threat to farmers' job security and could disrupt presently-stable sectors of the economy. The discussion becomes even more significant with the sort of general learning and skill-acquisition we target in this work.

On the other hand, it has been known that technological revolutions do spur growth. Some argue that we are living in the most prosperous age as humans [61]. As some jobs will be lost, new jobs will be created. However, it is without a doubt that those will be focused more and more on the strengths of humans, i.e. creativity and high-level cognition.

Education in this context becomes crucial, more than ever before. There could be jobs where humans work alongside capable robots, leveraging the strengths of both. In this regard, the case of the revised Tesla Gigafactory automation is an excellent example. By including more humans, not less, Tesla was able to ramp-up the production of the Model 3 car (and with it secure its economic survivability).

Furthermore, robots that are skillful and interactive can have many application domains to serve humans. Service robots, Therapeutic robots, Companion robots and Search & Rescue robots are but limited examples [62]–[65]. I believe that the full-spectrum of uses for smartphones was not conceived before the advent of the System-On-Chip (SOC) chips. That is, of course, it was conceived that having phones like powerful small computers would be nice, cool, useful and beneficial. After all, there were attempts at it in the 90s yielding products such as the Palm handheld devices. It was not until the iPhone, however, that the storm has been unleashed through the App Store. The iPhone itself was not the revolution, it was the apps built to run on it.

In this way, the robots we propose to build are not made for a specific purpose or application. In a sense, we are laying foundation for technologies that could replicate the role of the iPhone in robotics; a general platform capable of serving the multiple purposes and needs of its users. The potential impact it could have on society is significant. There is no doubt, either our method or others' will in time be successful. Robots will be a reality of our world, helping us with our everyday lives.

CONCLUSION

The method of Experiential Robot Learning was presented to work around the challenges of hand-coding as illustrated in the previous chapter. The Approach and Implementation components of the ERL method were provided in detail. We constructed a scale called Attractiveness Score to assess if an implementation meets our success criteria or not.

Related works were examined in the Literature Review section. We attempted to define our dimensions of uniqueness within the vast body of related work. Finally, a discourse on the broader impacts was presented.

The method of ERL needs to be validated experimentally through experimental studies. We perform such studies in the coming chapters. The studies will investigate the different aspects of using neural networks in the context of ERL, such as new training techniques and unique hybrid neural architectures. While these studies are not expected to solve completely the problem of robot learning, they should generate new insights and knowledge to pave the way and contribute towards a solution.

CHAPTER 3: TRAINING DNNS WITH DEEP LEARNING

EXPERIMENTAL STUDY 1: INTERACTIVE ACQUISITION OF VISUAL OBJECT DICTIONARY

INTRODUCTION

To demonstrate the applicability and scalability of ERL, we show that a robot can expand its knowledgebase of identifiable objects by combining human-coaching and fine-tuning, a well-known practice in training deep convolutional networks [6]. The top convolutional layers of a CNN encode the higher-level features specific to a class. Those layers can be re-trained, along with the Fully Connected layer(s) to refine classifications. For example, it is common practice to import a network pre-trained on ImageNet dataset such as GoogLeNet [66], modifying the top layers and tuning the network for a specific classification task.

FIGURE 7 NAO ROBOT

System Design

The system we use is as follows. A desktop computer, with an Nvidia TitanX GPU is used as a back-end. The computationally-demanding nature of Deep Learning imposes this constraint. The computations cannot be performed locally, on-board. As the field advances, resulting in more efficient development tools and as embedded hardware advance, perhaps this can be alleviated. The robotic platform of choice is a humanoid type called NAO. It is equipped with two cameras, one facing downwards and another facing forward. It also comes with a proprietary software package of Speech Recognition. This allows it to interact with a human coach in natural language.

The interaction occurs in the manner of a Finite State Machine (FSM), as shown in Figure 8. The robot transitions between the following states: *Rest*, *Observe Object*, *Inspect Object*, *Generate Predictions*, *Inquire the correct label* and *Fine-tune network*. The transitions occur in co-operation and communication with the human coach. For instance, if the robot is about to inspect the object, it first informs the accompanying human in speech. Similarly, when it generates a prediction for an object, i.e. classification, it informs the human in speech and inquires whether it was correct or not.

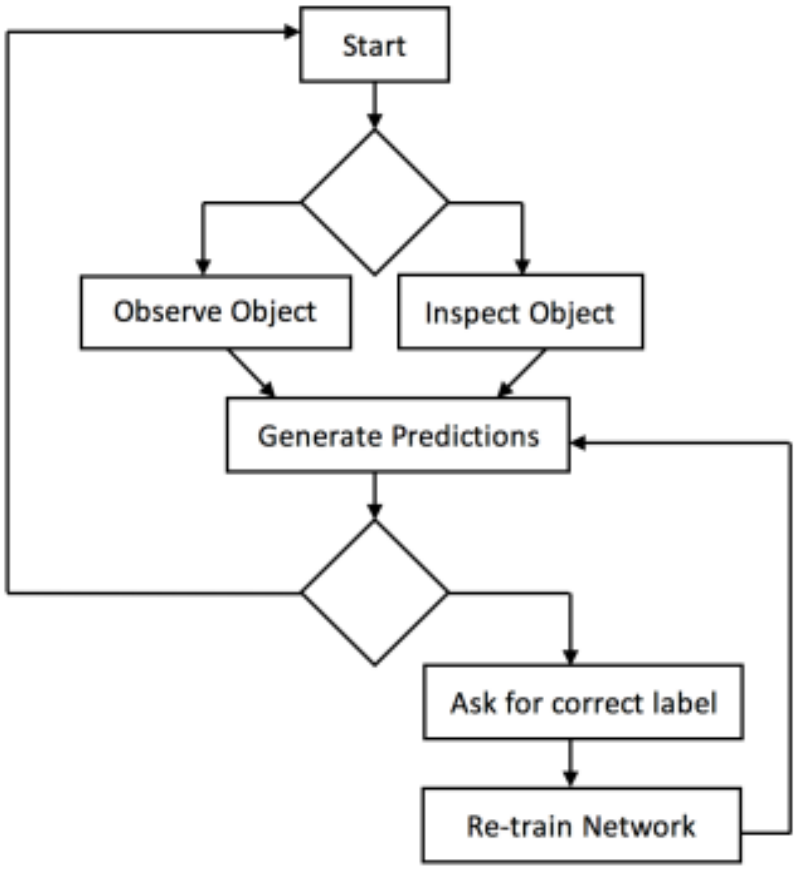


FIGURE 8 EXPERIMENT'S SYSTEM FLOWCHART. RECTANGLES REPRESENT PROCESSES. DIAMONDS REPRESENT DECISIONS. THE FIRST DECISION IS WHETHER TO USE OBSERVATION OR INSPECTION MODE. IN OBSERVATION MODE, THE HUMAN COACH PRESENTS THE OBJECT TO THE ROBOT'S CAMERA. IN INSPECTION MODE, THE ROBOT AUTONOMOUSLY INSPECTS THE OBJECT BY WALKING AROUND IT. THE SECOND DECISION IS WHETHER THE OBJECT WAS CORRECTLY IDENTIFIED OR NOT, DEPENDING ON WHICH THE SYSTEM EITHER LOOPS IN A RETRAINING PROCESS OR NOT.

NETWORK ARCHITECTURE

The neural network is written and trained using Keras, an API to Google's TensorFlow [67]. We first import a standard VGG16 [68] network, without the fully connected (FC) layers. Instead of the standard 4096 nodes in the last two layers, we use 64 and 32

respectively. The reason is that VGG16 is originally trained to recognize 1000 classes. It means that it has a quite large entropic capacity. If we used the network as is, it would be strongly capable of over-fitting due to the limited amount of data we train on, and the number of classes, 6, we are initially interested in. Therefore, the model needed to be constrained in such a way as to prevent over-fitting and still maintain an appropriate entropic capacity.

Classes

There is a noise class among the 6 classes the network is trained to recognize. It is inspired by the work of Bendale et al. [69]. We train it using images that do not belong to the other 5 classes we originally train the robot to identify. Care is taken that the size of the dataset in this noise class does not go beyond the size of the other classes. Otherwise, it would create an imbalance in the training. The reason we need a class of this kind is that since our objective is to create open-ended robotics, the agent must not only identify objects it is trained on, but also objects it isn't. There are other proposals and ideas to implement a noise/unknown class [69], [70].

The entire dataset is about 1000 images. The number of images is deliberately chosen to be low. This stems from the idea of open-learning. If the robot needs to learn a new class, it must do so with very little data, around 100 images or less. This is because we are concerned with local experiences shaping the learning behavior. For example, if the robot is introduced to the concept of desktop computer, i.e. a new class for this object is created, it is unlikely it will have many, many instances of such object to learn from. It will need to

internalize the concept, i.e. class, only with the available instance(s) of the object. As it experiences more instances of the same class, desktop computers in this example, it can refine its ability to identify them by engaging in the process we present.

However, we attempt to address this matter in two ways. The first is that the trainer in Keras can generate new training images based on the ones in the training dataset. These undergo random alterations such as rotation, zoom and flip. In that way, the model rarely sees the same image twice.

The other way we address the limited dataset is to increase the entropy in the set as much as possible. We do so in two ways. First, we can put the object directly in front of the robot's camera and rotate it. The camera takes many snapshots of the object. The idea of rotation comes from the desire to capture the different aspects of the object. For example, a stapler is not a symmetrical object, i.e. it looks different from the side or the top than it does from the front. Each view presents different features by which the object can be classified. Contrastingly, a ball is symmetrical all around, at least in shape.

The second way to increase entropy in the data set is to have the robot autonomously walk around the object and take snapshots, we call this inspection. The movements are pre-determined, however. We envision that as the robots get better in motion, possibly through future work applying ERL, the robot will generate its own motion sequences for this task. In those two ways, the agent accumulates rich data on which to train itself to recognize a (new) class. The motivation, again, is gradual, and continuous self-improvement.

Training and Fine-Tuning

The network is instantiated with VGG16 ImageNet weights. All the convolutional blocks are set to non-trainable, i.e. fixed weights, except for the last one. The hierarchical nature of Deep Learning means that the lower convolutional blocks detect abstract features such as Edges and Curves. Therefore, we are only interested in training the last convolutional block. Adadelta [16] optimizer is used, with a learning rate decay of 0.004. Again, it is just a design choice for our system. In building Neural Networks, there are many design decisions to be taken, depending on the performance constraints required for the model.

The fine-tuning process is as follows. If the robot is presented with an object it cannot label correctly, it asks the human coach whether to label it or not. If the human chooses to label the object, the robot will inquire for the correct label and either create a new class for the object or add it to an existing class. Immediately afterwards, the last convolutional block and the FC layers of the network are automatically re-trained on the new dataset. The robot then attempts to generate a label for the object, if correct, then the process ceases. If still incorrect, the robot gathers additional data and the network is retrained. The coach thus only provides a label, not training examples/data.

When generating a label, i.e. making a classification, the robot does not use one image. This is perhaps a significant divergence from traditional Image Classification tasks. Since the robot can navigate the environment, we can gather multiple images of the same object, from different angles. In doing so, we generate an aggregate label over 25 images in total. In such a way, if the classification is incorrect over multiple images, it still will not sway the judgement over the entire classification set. This is in contrast with producing

confidence scores in predictions for single images [70]. We can afford to use this approach because our implementation does not require a constraint on making a correct label for every single image.

We define an early-stopping mechanism to terminate fine- tuning. When the validation loss stops decreasing for 10 consecutive epochs, the training halts. The system then transitions to the second decision in Figure 8. It generates a new label and communicates it to the human, to test if it has learned the object or not.

Related Works

In this section, we present some works which are relevant to our context and highlight their intersections and differences.

Some works employ deep neural networks in developmental, physical robotic implementations [71]–[73]. However, the authors are more concerned with the respective implementation domain rather than a certain general methodology.

Perhaps the closest to tread a similar approach to ours is the work by Di Nuovo et al. [46]. Their research though mainly tackles the acquisition and manipulation of symbols and numbers; in a way that is also inspired by child development.

Furthermore, the work in [47] by Luo et al., adopts a similar mindset to ours. Their implementation addresses a different scope and problem to ours: that of determining which direction a robotic arm moves in using joint angles. They similarly address how to generate

motion sequences for the arm to move in a defined direction. They employ a variation of deep auto-encoders.

In the field of Reinforcement Learning (RL), Li et al. [48] present an approach to create open-ended intelligent robots. The authors advocate combining actor-critic models and reward-policy mapping. Their results are for a rudimentary navigation task, and only in simulation. The work in [49] also uses RL to have a physical humanoid acquire motor skills such as standing, leaning and walking. The humanoid, though, needs to experience a virtual environment in simulation first until it converges on a semi-reasonable policy. This is followed by a period of motor-babbling, until the humanoid refines its policy. Meola et al. [50] discuss using RL to generate different motion types in robots such as Rhythmic motion, inspired as well by motor development in children.

Sigaud et al. [5] present different DL techniques and discuss their applicability on developmental robotics. It is a primary foundation on which we build upon this experiment. The authors' presentation is only theoretical however, and no physical implementation is given.

Results and Discussion

In this section, we present the results of our implementation of the Experiential Robot Learning method. We choose to apply ERL to robot visual perception using deep Convolutional Neural Networks (CNNs). The classes and datasets are as outlined in the previous section. The system FSM handles the transition between states, communicating each step to the coach and receiving feedback, in speech.

There are two experiment sets. In the first set the human presents an object to the robot's camera. During a period of a few seconds, the robot takes snapshots while the human rotates the objects to present it from all angles. In this way, a classification batch is constructed. The robot then communicates the aggregate label generated for the batch. The human then coaches the robot, either by affirming its predictions or by engaging with it in the fine-tuning process as described earlier. We call these **Observation trials**. There are 3 trials, as shown in Table 6.

The second experiment set we call **Inspection trials**. Those are the first steps towards autonomous robot exploration. The robot leverages its physical body to move around and inspect an object. Somewhat like children when something captures their attention; they inspect it from different angles because they know they are looking at the same object. Similarly, the robot captures snapshots as it is moving and they all are aggregated to generate one label, i.e. they all belong to the object in question. The images are then run through the CNN and the coaching process continues as described earlier. The results are presented in Table 7.

TABLE 6 IN THIS TABLE, THE RESULTS OF THE OBSERVATION EXPERIMENTS ARE GIVEN. A HUMAN PRESENTS THE OBJECT TO NAO'S CAMERA. THE HUMAN COACH ROTATES THE OBJECT AS THE ROBOT IS TAKING SNAPSHOTS TO SEE IT FROM ALL ANGLES.

Trial	**1**	**2**	**3**
Object Presented	Ball	NAO robot	Paper plate
Previously Known/Unknown	Known	Unknown	Unknown
Correct classification	Yes	No	No
Fine-tuning required	No	Yes	Yes
Correct classification after fine-tuning	N/A	Yes	Yes

TABLE 7 IN THIS TABLE, THE RESULTS OF THE INSPECTION EXPERIMENTS ARE GIVEN. THE ROBOT AUTONOMOUSLY WALKS AROUND THE OBJECT WHILE TAKING SNAPSHOTS, TO SEE IT FROM ALL ANGLES.

Trial	**1**	**2**	**3**
Object Presented	Ball	Wallet	Bottle
Previously Known/Unknown	Known	Unknown	Unknown
Correct classification	Yes	Yes	No
Fine-tuning required	No	No	Yes
Correct classification after fine-tuning	N/A	N/A	Yes

Since our work addresses partially the open set recognition problem, we need to use objects that the robot, i.e. CNN, is not trained on. This is what is conveyed in the Known/Unknown row, whether the robot was trained to recognize the outlined object or not. In this way, we test the noise class presented in the previous section. From Table 6 and Table 7, it can be inferred that the noise class performs satisfactorily and can be enhanced through the coaching process.

Through our trials, we have tested every possible branch in the system. First, we test if the robot can classify objects on which it was trained. That is seen in trial 1, in both tables. The robot is presented with a ball and it classifies correctly so there’s no need for fine-tuning.

Afterwards, we test if the robot can recognize that the presented object is not part of its Known classes; this is a test of the quality of the noise class. If the robot misidentifies an object, then there are two procedures. The first procedure is to create a new class for the object and re-train the network to classify instances of that new class. This is the case in observation trial number 2. An alternative procedure is to enhance an existing class by adding the classification batch to the training data and fine-tuning the network. This is the

case for observation trial number 3 and inspection trial number 3. This fine-tuning procedure occurs in a loop until the robot correctly assigns the object to its class.

The retraining process takes relatively little time, under 4 minutes and 50 epochs. It means that the robot can engage its human coach and provide & receive feedback in a timely manner; especially since they interact in natural language.

Attractiveness Score Assessment

In this section we assess the robot's performance through the Attractiveness Score measure. The assessment is provided in Table 8.

TABLE 8 THE ATTRACTIVENESS SCORE MEASURE FOR THE EXPERIMENT

Criterion	Assessment	Score
Learning	Robot demonstrates learning	5
Convergence	Robot learns in 1-2 trials	1
Pre-training	Pre-training takes less than 24 hours	1
Dataset Compilation	Dataset takes more than 24 hours to compile	0
Portability	Model not deployable on embedded systems	0

The total score of the robot is 7/10. According to the AS measure, this experiment has passed our success criteria.

Experiment Conclusions

In the preceding sections, we have presented a development method for open-ended, developmental robotics based on Deep Learning techniques called Experiential Robot Learning (ERL). It is inspired by observations on how children learn and enabled by learning directly from sensory input, through DL. To the best of our knowledge, we are the first to propose this method in the scope & context defined herein.

To evaluate our method, we applied it to the problem of open-world robot visual perception using deep convolutional networks. The resulting network architecture featured a robust noise class, since we are addressing open-set classification. We hope other researchers can benefit from the detailing of our architecture, as well as the training procedure.

The results indicate that the robot is capable, through experience and coaching, to expand its visual vocabulary of identifiable objects.

The aim is to create robots capable of learning from raw sensory input by being immersed in the real-world environment. This eliminates the need to engineer features, and gives the robot potential to be a self-improving, self-propelling entity. Ultimately this is our vision for robots. The application of human coaching in this experiment is only periphery. We also recognize the limitation in scalability a coach poses for this system. As Deep Learning practices advance, particularly Unsupervised Learning, we should aim to develop systems following ERL approach to leverage them.

Limitations of Deep Learning

The previous experiment was a success. It has met our success criteria, and the system design adhered to the ERL method. The learning would go on autonomously but would require human-coaching should we want the robot to know the common English names for the objects. Should we alleviate this requirement, it is quite easy for the robot to create its own names, i.e. labels, for the objects it encounters. Though, of course, such names as it will create will not conform to the English names.

Despite not being concerned with catastrophic forgetting, as we mentioned in an earlier section, the representational capacity of the model can be an issue. The robot would compile larger and larger datasets, requiring re-training. As the dataset becomes larger, the ability of the model to distinguish all those data points (belonging to different classes) will diminish because its representational capacity has remained constant. It is possible however to engineer the learning system to autonomously increase its own model's parameters, and consequently increase the representational capacity.

Despite these successes, we found that this approach had several limitations. First, the type of task that the robot can be trained to perform (or be able to learn) must be a "differentiable" task. Or more specifically, the task has to have a representative loss function that is differentiable.

Let's dive a little into what a loss function is, and what it is supposed to accomplish and how it is supposed to behave. A loss function is a function that computes a value given the neural network's output. By optimizing the loss function, i.e. find its optimum, the

optimizer is supposed to “train” the neural network. The optimizer thus accomplishes the task at hand by optimizing the loss function. The loss function should give high value if the network’s output does not solve or aid in solving the task. Conversely, if the network’s output solves the task or is helpful, then the loss function should compute a low value. In that way, the loss function reflects the overall quality of the network’s outputs in relation with the task being accomplished. High loss indicates optimizer failure and low success.

In this context, a loss function represents the task. If the loss function is unrepresentative of the task, then optimizing the neural network according to it will not aid in accomplishing the task. Unrepresentative means that getting a low loss value from the loss function doesn’t actually translate to successfully performing the task at hand. One can easily conceive many scenarios/tasks where success is not easily measured through a loss function. Learning any task with Deep Learning requires a loss function. Note that a loss function can incorporate labels, such as in cross-entropy, or it can be label-free such as in mean-squared error for autoencoders.

In addition to the loss function being representative of the task, it also needs to be differentiable. The loss function cannot be non-smooth or have discontinuities. By extension, the network architecture must also be end-to-end differentiable. Moreover, it is assumed that the hypersurface created by the loss function in the space of network weights is convex. If the loss hypersurface is not convex then the optimizer will not converge on a proper solution. Recall that gradient-based optimization algorithms are used to train the neural network, which are part of the stochastic convex optimization family. The gradient-

following behavior of those algorithms will not be useful if the function being optimized is non-convex. There are many functions whose topologies are not convex. The gradients of those functions are thus "uninformative", i.e. it does not lead to a global optimum. Those are some of the challenges associated with derivative-based optimization, aka. Deep Learning, based on the use of loss functions.

These challenges may not be prominent in the field of Computer Vision tasks. For robotic tasks, however, those challenges are pronounced. The field of Deep Reinforcement Learning (DRL) tackles many tasks of similar structure to what we would assign our robot. The tasks are usually performed in a non-stationary environment, where the agent's actions carry some effect on the subsequent possible state spaces. In the field of DRL, it has been noted that gradients are not always informative. This remark has been cited by OpenAI researchers in [74]. We highlight a quote from their paper "*…a large source of difficulty in RL stems from the lack of informative gradients of policy performance*". The authors introduce the use of Evolution Strategies instead of gradient-based methods to solve RL tasks. Similarly, Uber AI researchers in [75] propose using Genetic Algorithms to solve RL tasks. Both have been found competitive to gradient-based methods.

In our experiences, while the previous experiment has been a relative success the subsequent attempts to design systems according to ERL have failed. These failures, we hypothesize, stemmed from our use of gradient-based optimization. The distinct lack of exploration of the action space can be associated with the greedy nature of gradient-based algorithms that are primarily geared towards gradient-following behaviors, i.e.

exploitative. The challenges associated with designing a proper loss function, as we just discussed, for the tasks we wanted the robot to perform may also stem from the use of gradient-based optimization. For those reasons, we switched directions and veered into the realm of derivative-free optimization algorithms to train our neural networks. Limitations are summarized in Table 9.

In doing so, we expanded our definition of ERL implementation. It is thus not about modern "Deep Learning" architectures. ERL implementation is about malleable structures that are capable of representation such as deep neural networks, and the ability to train said structures. Conventional deep neural networks are but one example of malleable representational structure, there are others such as Extreme Learning Machines. We choose deep nets for their relevance to today's technology, academic and industrial applications and interests. However, the black box optimization methods developed henceforth should be usable with minor adjustments for other learning structures.

This switch from derivative-based to derivative-free optimization provides other affordances. Training in low-precision can be easier than with gradient-based methods. Training itself is simpler, in general, since only weight updates are performed through perturbation and backpropagation is eliminated. Non-differentiable elements in the network architecture can now be incorporated. Even the reward policies can be simpler to define, as we shall see in the coming chapters.

CONCLUSION

Let's review this chapter. Starting off with the principles of the ERL method, we applied them to design a robotic system capable of autonomous learning. The learning was in the form of updating and improving upon a visual dictionary of real-world objects in real-time from real-world data gathered through interaction with the real-world physical environment. The robot was successful and achieved a high score on the AS scale introduced in the previous chapter.

TABLE 9 SUMMARY OF DEEP LEARNING LIMITATIONS

Item	Limitation
Loss Function	Representative, Differentiable, Convex
Natural Behavior	Greedy, exploitative
Gradients	Must be informative
Precision	High-precision required
Neural Architecture	End-to-end differentiable

However, further attempts to extend this style of learning were not successful. Many challenges were encountered and chief among them was the non-exploratory behavior of the robot and the design of a proper loss function. Since we need to adhere to the ERL method, it was deemed that the derivative-based optimization approach was the culprit and will be exchanged for derivative-free optimization. The switch to DFO allows previously inaccessible behaviors and designs for the system. We summarized the limitations posed by Deep Learning in Table 9 for easy reference.

Chapter 4: Training DNNs With Evolutionary Algorithms

Experimental Study 2: From Playing Flappy Bird to Robot Learning through Accelerated Neuroevolution

Introduction

The algorithms of Stochastic Gradient Descent (SGD) and Backpropagation (BP) were invented decades ago [76], [77] [78]. There have been many variants that improve the performance of SGD such as the widely adopted ADAM algorithm [17]. Those algorithms gave rise to what is now dubbed Deep Learning. In effect, Deep Learning can generally be considered as gradient-based optimization of deep neural networks.

Gradient-based methods, however, pose constraints such as end-to-end differentiability of the model. When dealing with some tasks, for example Hard Attention [79], this differentiability requirement can become a limitation. In addition, when one explores the field of Reinforcement Learning, the constraints are more pronounced. Agents suffer from lack of exploration and stagnation due to the natural behavior of SGD and uninformative gradient signals [80]. There has been many techniques to get around those limitations in Computer Vision [81] and Reinforcement Learning [82].

In this experiment, we propose accumulations to the Experiential Robot Learning method based on the observations concerning gradient-based optimization. Primarily, we propose the usage of Deep Neuroevolution as an alternative optimization technique.

Evolutionary algorithms have been notoriously known for their convergence problem (they usually take a relatively long time to converge on a solution). There are multiple attempts to combat this, most notably is the NEAT algorithm [83], and its variants such as FS-NEAT [84]. However, those approaches rely on mutating the network structure alongside weights. In addition, due to the large number of generations and populations within each generation, the number of physical implementations of Evolutionary Algorithms has thus far been limited [85]. This is especially pronounced when considering that ERL requires online learning, though it does allow a bootstrapping process to precede. In this context, we propose an Accelerated Neuroevolution, or ANv1, algorithm that improves upon a baseline algorithm in the game Flappy Bird. Our algorithm is engineered to display several desirable behaviors such as faster convergence, stability and adaptive exploration. We deemed ANv1 suited for physical implementation and tested it on a NAO robot in a visuomotor learning task. The goal is to center an object in the robot's field of vision. A visual reference of the task is given in Figure 10.

In addition, we introduce a gradient-trained CNN that classifies the location of an object. The CNN is seamlessly integrated with an evolved network, to create a hybrid system of gradient and gradient-free trained networks. ANv1 performs remarkably well in a physical, although simplified, environment.

Experimental Setup

We chose the algorithm in [86] as a baseline for our experiments. Briefly, the Batchu algorithm exposes a neural network to an evolutionary process through favorable breeding of higher-ranking populations according to a fitness metric. In the Flappy Bird game, the fitness metric is the performance of the agent playing the game, i.e. avoiding obstacles. The Batchu algorithm features a coarse notion of crossover, the offspring's networks are composed of entirely swapped layers of weights. That is, a layer's weights, incoming or outgoing, are entirely inherited from one of the parents. A random mutation is then applied to the weights, with a selection probability of 0.15. The self-reported performance in of the algorithm is about 1 hour, i.e. hundreds of generations, of training for convergence on a capable agent [86]. This result corresponds to a fully-connected network with a hidden layer of 7 neurons, and a population size of 100 agents for each generation.

Accelerated Neuroevolution v1 (ANv1)

The proposed algorithm, Accelerated Neuroevolution v1 or ANv1, needs to be much more efficient if it will be used for physical implementation. This may come at a cost of reduced exploration behavior. We modified a few behaviors of the baseline algorithm as such. The algorithm outline is given in Table 10.

TABLE 10 OUTLINE OF THE ACCELERATED NEUROEVOLUTION (ANV1) ALGORITHM

Stage	**Description**
Assemble new generation	- Assemble generations of N populations
Perform the task	- Perform the assigned task, the populations experience the environment either simultaneously or separately
Evaluate Performance	- Evaluate the performance of each population according to the fitness metric - Determine the Winner as the population with the highest score - Determine the Mutation Resistance rate based on the calculated fitness metric of the Winner (either reset to the original value of 95 or lower by 0.05)
Selective Breeding	- Preserve the Winner - Perform Crossover for the Royal Family - Perform Crossover for the rest of the populations
Random Mutations	- Preserve the Winner - Perform mutation for all other populations using the newly-determined Mutation Resistance rate

First, in Crossover instead of taking an entire weights layer from a parent, the weights of the offspring are taken in a crisscross, i.e. checkered, pattern from both parents. There are two offspring from this process, depending on which parent we start with. Thus, if the first weight belongs to parent 1 then the second weight will belong to parent 2, and so on. Our intuition was that this equal participation allows the inheritance of both good and bad genes from the parents. In addition, it may encourage sparsity in the model.

Second, we designed an adaptive mutation scheme to balance perturbation, i.e. exploration, and stability. The scheme randomly mutates weights with a 5% selection probability. We call the inverse of this selection probability, i.e. 95%, Mutation Resistance rate. The rate is dynamically adjusted by comparing generational entropy of the fitness metric. For each generation after the first, the fitness metric is examined and compared with its predecessor.

If the score is found to lie within a pre-defined interval, e.g. -5-10%, then it is dubbed stagnant. If there is a stagnant generation, the Mutation Resistance is accumulatively lowered by 5% for the upcoming generation. Each stagnant generation thus experiences a higher probability of mutations. If finally, one generation's fitness metric is sufficiently higher or lower than the stagnation interval, then the Mutation Resistance is reset to 95%. We wanted to balance exploration and convergence. The algorithm needed to be able to get out of local minima, and our intuition was that this behavior can help with that. Note that the magnitude of the mutation is constant, it is the mutation selection probability that changes adaptively.

Third, we made amendments to help increase convergence rates. In our experience the baseline algorithm did not exploit breakthroughs greedily enough. If for example one of the populations was able to demonstrate a better behavior than the others, its good genes are usually lost in the crossover and mutation process. Therefore, we introduced a Winner and Royal Family concept (Elitism and Elite population).

In the iterative search of evolutionary algorithms, some populations (aka. samples) perform better than others. Using the mechanisms of mutation and crossover, the parameter vector is altered to create a pool of new, unique samples. This pool is then re-evaluated, and the fitness (aka. Score) of each sample is The Winner of each generation is the best-performing agent. This Winner is preserved to the following generation, without experiencing mutation. This preserves the best-performing agent in hope to retain its attractive properties. The Royal Family is the winner breeding (performing crossover) with other

previous winners (or itself if in the first generation), and undergoing mutations. The size of the Royal Family is predefined at the start of the algorithm. We typically choose a low number such as 4. Finally, the rest of the populations breeds with the Winner and undergoes mutations as normal. In this way, the Winner's good genes are dissipated to the entire upcoming generation, with various concentrations. The preservation of the Winner agent ensures this unique parameter vector endures without mutation. The Royal Family population aims to improve the Winner's parameter vector by exposing it to mutations. Breeding the rest of the population with the Winner rather than amongst themselves aims to create radically new parameter vectors based on the Winner's while allowing the inheritance of bad genes. We view bad genes as necessary to create new behaviors and encourage entropy within the genetic pool. In essence, with those choices we attempt to maximally capitalize on the good performance of any particular agent.

LOCATION NET (LOCNET)

We developed our algorithm described in the previous section to perform in the physical environment. The Flappy Bird game is simply a demonstration of its behavior in a simulated environment. The task we chose for the first implementation is the centering of a static object in the robot's field of vision. It is a steppingstone for more complex physical learning tasks such as real-time moving object tracking.

FIGURE 9 TRAINING SAMPLES FOR DIFFERENT CLASSES OF THE LOCATION NET CNN. NOTE THAT (A) AND (C) SHOW THE SAME OBJECT BUT IN DIFFERENT LOCATIONS, WE CALL THIS AN ADVERSARIAL SAMPLE. THE TRAINING SET CONTAINS MANY SUCH SAMPLES. THE INTUITION IN DOING THIS, IS THAT IT ENCOURAGES THE NETWORK TO FOCUS ON THE LOCATIONS OF THE OBJECT RATHER THAN THE OBJECT ITSELF.

We trained a CNN to classify an object's location in the robot's field of view. The network divides the visual field into a 3x3 grid, a class for each section. In addition, there's a Null class if there's no object detected in the image. Namely, the 10 classes are: Top Left, Top Center, Top Right, Left, Center, Right, Bottom Left, Bottom Center, Bottom Right and No Image. Location Net is designed to center any object regardless of what it is. The choice of using a sign, as shown in Figure 10, is only for convenience. The sign can be replaced with any other object. The network is based on a modified VGG16 architecture. Specifically, the fully-connected layers are shrunk to accommodate the relatively simple classification task. It is trained on a manually-assembled dataset of objects against a plain background in various locations in the 3x3 grid.

In total, the training and validation sets contain over 1,500 images. The training and validation sets feature adversarial examples by having the exact same object and the exact same background but in different locations. Intuitively this encourages the classifier to

recognize the locations rather than the objects in an image. Samples of the training and validation sets are given in Figure 9. Location Net, or LocNet for short, was trained with the ADAM optimizer and written in Keras with a TensorFlow backend. It is a gradient-trained network. Training takes less than 10 minutes on an Nvidia Titan Xp GPU. The network achieves over 90% accuracy for both the training and validation sets after 35 epochs. We deem this sufficient accuracy for the given task. Our task is neither time-sensitive nor action-critical, hence we can tolerate this error. The network can correct misclassifications through the redundancy of a video stream.

OBJECT-CENTERING TASK

In this task, the robot centers a static object in its field of vision by moving its head left or right. The robot's head is controlled by an evolved neural net. This network is seamlessly stacked under the Location Net CNN, described in the previous section. That is, the evolved net takes as input the output vector of the CNN and maps that to a command. The control net issues one of three commands: *Move Left*, *No Movement* and *Move Right*. Hence, we have a discrete control scheme, with a single Degree-of-Freedom, DOF.

FIGURE 10 THE NAO ROBOT LEARNS TO VISUALLY CENTER THE OBJECT (THE PAPER SIGN) BY MOVING ITS HEAD.

Over the generations, the evolved control net has to figure the mapping of the classification to the correct control action. For example, if the CNN classifies the object to be on the ’Left’, then the net has to issue a ’Move Left’ command. The robot then moves shifts its Head Yaw angle to the Left by a fixed amount. Once the object is classified as ’Center’, the net has to issue ’No Movement’ commands to ensure stability. This mapping is reflected by reward assignment. For every correct action the network’s total reward is incremented by 100. At the end of the 5 episodes, the total reward reflects the network’s performance. This total reward is our ’fitness metric’, through which the network is evolved. Intuitively, this reward assignment is what guides the ANv1 algorithm, i.e. Robot’s, behavior. If we want a different behavior, all we need is to change the reward assignment scheme.

COMBINED SETUP

In the following section we outline the experimental setup used to produce the presented results. In the first set of experiments two algorithms play the game of Flappy Bird on a desktop workstation. We use an Nvidia Titan Xp GPU to calculate and implement the populations, i.e. neural networks. The networks are written in Keras and implemented in TensorFlow.

For the second set of experiments, we use a NAO robot to center a physical static object. The algorithm runs on a desktop workstation with an Nvidia Titan Xp GPU. The robot captures its current view using its front camera and sends it to the workstation over a wireless local network. The CNN accepts the image as input then each control network, i.e. population, gets to control the robot for 5 steps. The goal is to center the object within those 5 steps and ensure stable convergence. Each 'Move' command increments the current position of the Head Yaw joint angle by a fixed amount. This amount is small enough to allow fine control, and limit overshooting. It is empirically chosen, based on observation. Incrementation is implemented by querying the robot for the current joint angle, and adjusting it based on the neural network's choice. Finally, the new position is sent to the robot as a command over a wireless local network.

Results

Flappy Bird Video Game

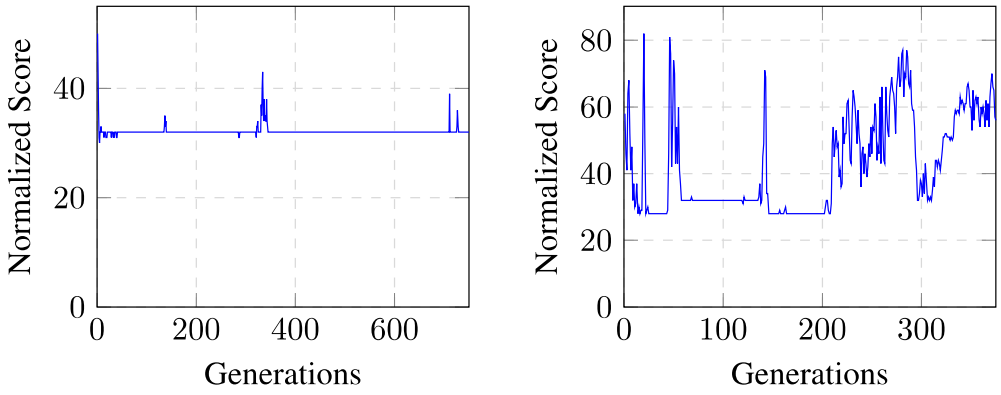
In this section we present the results of playing the Flappy Bird video game both with the baseline algorithm and our own algorithm. Note that the game has no end. That is, an agent capable of playing the game, can potentially go on indefinitely. This means there is no

definite metric for convergence. For that reason, we qualitatively pronounce whether an agent can play the game based on our own observations.

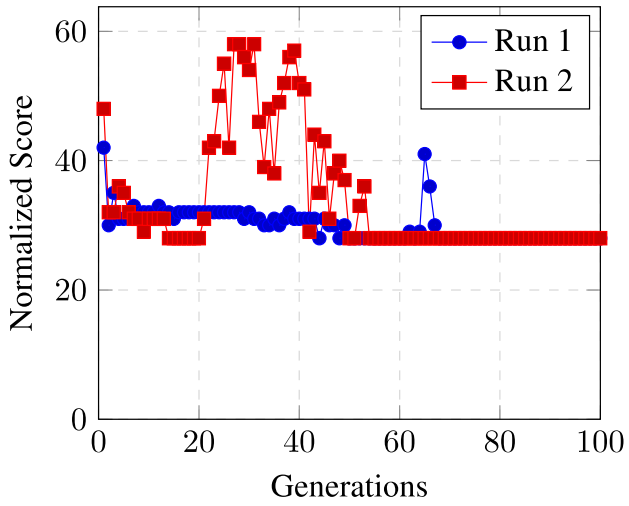
However, we understand that qualitative assessment is not a robust benchmark to compare performance. Thus, we report the generational score, normalized for population size, over the number of generations as a means to compare the algorithms. Normalized generational score is the aggregate score for the entire generation divided by the number of populations. This gives us an idea of how the entire generation performs rather than an individual outlier. In addition, we chose the number of generations, i.e. rollouts, as measure instead of time, because time may vary across tasks and domains. The number of rollouts, on the other hand, reflects the performance of the algorithm.

As a reminder, one of the motivations of creating our own algorithm is that evolutionary algorithms are generally known to be unfit for implementation on physical agents. This is due to their slow convergence and impracticality considering the typical population size.

The baseline neuroevolution algorithm by Batchu, described in the previous section, is reported to train agents capable of playing the game after roughly an hour of training, i.e. over a thousand generations. The algorithm operates on a single-hidden layer network of 7 neurons. It takes as input a tuple of 3 numbers representing the distance to the nearest pipe, the gap height and the height of the agent.



(a) 750 Generations. (b) 375 Generations.



(c) 100 Generations.

FIGURE 11 DIFFERENT, INDIVIDUAL ROLLOUTS OF VARYING LENGTHS OF THE BASELINE ALGORITHM PLAYING FLAPPY BIRD. THE CHART REFLECTS POOR PERFORMANCE DUE TO THE LARGE NUMBER OF GENERATIONS NECESSARY TO ACHIEVE LEARNING AS REFLECTED BY A HIGHER SCORE. THE ALGORITHM FAILS TO CAPITALIZE ON BREAKTHROUGH GENERATIONS BY RELAPSING TO A POOR PERFORMANCE.

The self-reported results as well as those reported in Figure 11 for the baseline algorithm are for a population size of 50. That is, the algorithm is training 50 populations every

generation. As the number of agents increase the chance that a population has good genes increases. This in turn leads to quicker convergence, intuitively. However, requiring a large number of populations for convergence renders an algorithm infeasible for physical implementation.

In Figure 11 we present the results of using the Batchu algorithm over rollouts of varied lengths. It performed rather poorly. Notably, convergence is extremely slow, and the algorithm fails to capitalize on breakthrough generations. Ideally, we want learnings to compound over generations for quick convergence. In addition, we also see how sensitive it is to initialization. Each plot in the figure is a separate run. Performance varies drastically between (a) and (b) for example. From Run 2 in (c) we highlight how performance regresses eventually despite a promising start.

The ANv1 algorithm we presented and described in the previous section, operates in this experiment on a single-hidden layer network of 50 neurons. We increase the number of neurons by 7X from the baseline algorithm. In practical implementations, problems are not going to be as simple as Flappy Bird. We wanted to test how our algorithm responds to a larger network. We use the same population size of 50.

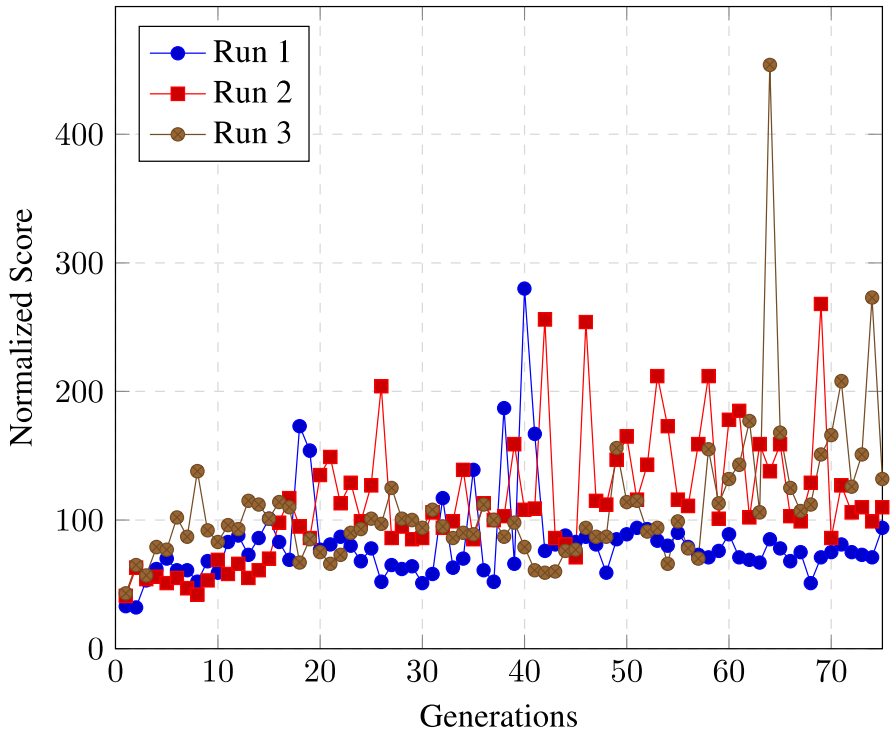


FIGURE 12 PERFORMANCE OF THE ANV1 ALGORITHM. THE ALGORITHM PERFORMS AT LEAST 1.7X BETTER THAN BASELINE (PRESENTED IN THE PREVIOUS FIGURE), OVER A MUCH LOWER NUMBER OF GENERATIONS, WHILE EVOLVING A NETWORK THAT IS 7X LARGER. THE ALGORITHM GENERALLY CAPITALIZES ON BREAKTHROUGH GENERATIONS, EVEN IF SUBSEQUENT GENERATIONS ARE NOT AS WELL-PERFORMING.

The performance of the ANv1 algorithm is presented in Figure 12. The algorithm is reported over only 75 generations. As the agents get better at playing the game, it takes longer for them to commit a mistake. Training thus becomes a diminishing returns factor of improvement. As we can see, the algorithm performs much better than baseline on average, an improvement of between 1.7X and 7.1X. Of course, those improvements arrive in a much shorter span of generations.

We can see behaviors in Figure 12 such as getting out of local minima by the accelerated compounding of score in the early stages. The dynamism of ANv1 is also highlighted by how the score doesn't plateau for more than a handful of generations. This behavior was engineered into the algorithm in order to achieve higher scores through exploration. We also note how the algorithm almost always performs better after a breakthrough generation, than pre-breakthrough. It shows that the algorithm capitalizes on breakthrough generations. This also an intended behavior that has been engineered. Stability and robustness can be highlighted by how the algorithm does not regress or backtrack to poor, initial performance. This is subtly different from capitalization on breakthrough generations.

Robot Object-Centering Task

This is arguably the most important section of the experiment. We deemed the ANv1 algorithm ready for implementation on a physical system. After all, it was our main motivation in pursuing this thread of inquiry. We decided to shrink the population size to only 15 in each generation. This number lowers the exploration potential of the algorithm but makes it far more practical. The network is still a single-hidden layer network, but of 10 neurons. This width is lower than the 50 neurons used in the earlier section. We chose this as the task is relatively simple and shouldn't require a network of larger representational capacity.

The task is described in the previous section. To reiterate, the robot is required to 'center' a static object by moving its head left and right. For each generation, i.e. optimization iteration, there are 15 populations. Each population controls the robot for 5 episodes. To

ensure robustness, we start the robot in 3 different locations. Specifically, we place the robot to the Left, Center and Right, of the object. For each location we carry our experiment 3 times. This is a total of 9 experiments.

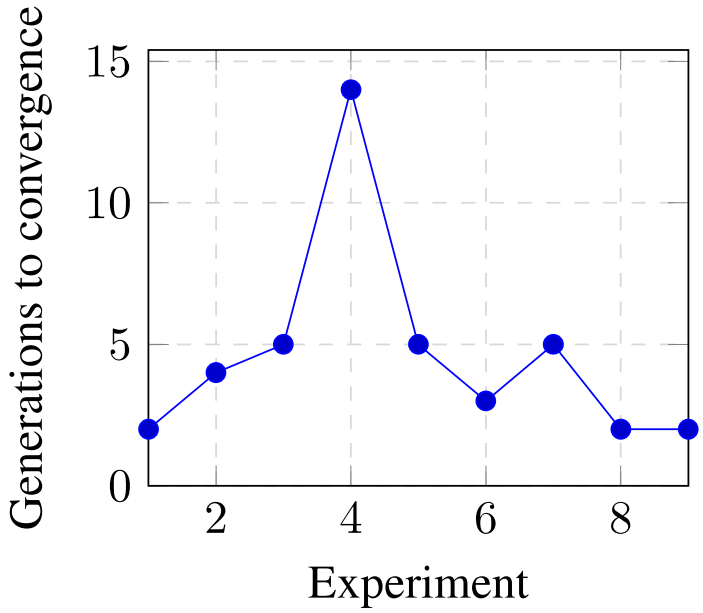


FIGURE 13 PERFORMANCE OF THE ANV1 ALGORITHM IN THE STATIC OBJECT CENTERING TASK. WE RAN 9 EXPERIMENTS, THREE FOR EACH POSSIBLE STARTING LOCATION, LEFT, RIGHT AND CENTER. THE ALGORITHM TERMINATES WHEN 7 OUT OF THE 15 POPULATIONS COLLECT MAXIMUM REWARDS. THE ALGORIITHM CONVERGES IN LESS THAN 5 GENERATIONS, WITH ONE EXCEPTION.

The algorithm stops when 7 of the 15 populations have received maximum rewards. That is, 7 or more populations successfully map the CNN classifications to the correct commands for all the 5 episodes, i.e. optimal agents. Once this threshold is achieved or passed, the algorithm terminates.

We have found that the ANv1 algorithm successfully navigates the task 9 out of 9 times. Each experiment generally took 10 minutes or less, with one exception. The results are shown in Figure 13. Experiment 4 in Figure 13 took 14 generations to convergence while the other two experiments in its class, 5 and 6, took 5 and 3 generations respectively. We assign this behavior to sensitivity to initialization. To a lesser extent, conservative mutations may also be a contributor. In ANv1, the initial mutation resistance rate, as outlined in Table 10, is 95%; and it decreases only in 5% decrements per generation. In general, we are satisfied that our choice of running 3 experiments per location has been rewarded in revealing this behavior.

Attractiveness Score Assessment

In this subsection we assess the performance of the robot on the ERL Attractiveness Score (AS) scale. The results are given below.

TABLE 11 ASSESSMENT OF EXPERIMENT ON AS SCALE

Criterion	Assessment	Score
Learning	Robot demonstrates learning	5
Convergence	Robot takes more than 10 trials	0
Pre-training	Pre-training takes less than 24 hours	1
Dataset Compilation	Dataset takes less than 24 hours to compile	1
Portability	Model not deployable on embedded systems	0

The robot thus achieves a score of 7/10. This score surpasses our criteria for success, and thus we consider this experiment a successful demonstration of ERL.

Conclusions

Motivated and guided by the Experiential Robot Learning, ERL for short, method, we proposed the Accelerated Neuroevolution v1, ANv1 for short, algorithm. We tested the algorithm in two scenarios. The first experimental set is in a simulated environment, playing the game of Flappy Bird. We found that the algorithm performs much better, 1.7-7X, than a baseline algorithm, while controlling a network that is 7X larger. We observed encouraging behavior such as capitalization on breakthrough generations, ability to get out of local minima, high dynamism while maintaining robustness and stability.

The second experimental set took place in the physical environment on a NAO robot. The goal was to center an object in the robot's field of vision. We developed a CNN called Location Net to classify where the object is. We found that the algorithm generally converges in a relatively low number of generations, 5 or less. We did observe, however, that the algorithm is sensitive to initialization. This is noted in the case of experiment 4 in Figure 13 where the initialization was likely adversarial.

While not reported here, we also observed how the algorithm makes heavy use of the adaptive mutation resistance rate. Whenever it was 'stuck' or plateauing, the algorithm would decrease the resistance rate until it got out of the flat region. We observed this for both the simulated and the physical experimental sets.

We recognize the limitations of our work, as well. The algorithm needs to be benchmarked on larger, more complex tasks. In addition, more realistic tasks will require evolving neural networks of greater representational capacities, i.e. deeper, wider and more varied.

However, a single DOF task is not unheard of in the domains of Robotics and Reinforcement Learning. Furthermore, given that this is an initial step, we believe it is a significant proposition and validation.

For future iterations, we aim to implement adaptive mutation magnitudes. While the mutation resistance rate is adaptive, the magnitude of the mutations is static. We think this will become useful when evolving more complex networks, whose behavior needs to be fine-tuned.

In conclusion, we presented a first foray into Accelerated Neuroevolution with a simulated and a physical learning experiment. The algorithm performed well in the assigned tasks and demonstrated encouraging behaviors. We like how a gradient-trained network, i.e. LocNet, was seamlessly integrated with an evolved network, i.e. the control net. We see this sort of integration proliferating into more applications that prove that a hybrid approach is needed.

Experimental Study 3: Multiple Search Neuroevolution

Introduction

In recent years, deep neural networks have been employed for tasks in various domains like Object Detection [87] , Robotic Grasping [88] and Machine Translation [89]. They are quite popular and powerful given their representational capacity and automatic feature extraction. To train those networks the standard algorithms used are Gradient Descent [90] variations such as Stochastic Gradient Descent (SGD) [76] and ADAM [17], all employing Backpropagation [78] [77]. Remarkably, Gradient Descent, SGD and Backpropagation were reported in 1847, 1951 and 1974 respectively. Since then, many techniques and variations have been reported over the years such as ADAM and Batch Normalization [91] in 2015.

In general, gradient-based algorithms are preferred when training neural networks because of their ability to handle complexities introduced by networks with millions of weights. However, there are also limitations when using a gradient-based optimization algorithm such as SGD. The neural network architecture and the loss function must be end-to-end differentiable. Consequently, there are several problems that cannot be directly modeled or addressed without some alterations such as Formal Logic, Discrete Action and Hard Attention [81], [92]. Another limitation is the lack of exploration due to the greedy nature of the algorithm, i.e. gradient-following. This makes the algorithm somewhat linear in the discovered solutions due to little exploration of the search space. For tasks that require exploration, this can make the training process challenging. Furthermore, the algorithm is

prone to getting stuck in local minima and saddle points. While there are remedies to such challenges, e.g. using Dropout, they may not always produce the desired effect.

For those reasons, we ventured to investigate derivative-free optimization algorithms that can potentially train deep neural networks. Optimization algorithms generally are divided into two categories, exact and heuristic [93]. Due to the combinatorial complexities of training neural networks, heuristics are almost always used. They are neither guaranteed to converge nor reach an optimal solution. Heuristics can further be divided into two categories, single-solution and population-based. Otherwise, there are many more lines to distinguish different families of algorithms by. Single-solution heuristics, also called trajectory methods, attempt to iteratively improve upon a single candidate solution. Single-solution heuristics are generally exploitation-based algorithms [94]. On the other hand, population-based heuristics attempt to improve a "population" (set) of solutions based on their "fitness" (performance) according to an objective function, over generations (optimization steps). Population-based heuristics are generally exploration-based. For those algorithms to achieve a solution, there must be some differentiator between the populations throughout the optimization process.

Specially designed for neural networks are Neuroevolution algorithms, a derivative of evolutionary computation algorithms. In 1994 Ronald and Schoenauer first reported using genetic algorithms to optimize a simple neural network for a toy task [95]. Since then there were many implementations and variations on Neuroevolution, such as NEAT in 2002 [83]. In general, Neuroevolution algorithms can be split into two categories. The first, such

as in NEAT, attempts to evolve the topology along with the weights of the network. The second attempts to evolve only the weights, such as in [75]. Since evolving the topology would add another aspect of complexity, we are only concerned with evolving the weights of the network. In addition, evolving the network topology does not allow a somewhat direct comparison with SGD.

To that end, this experiment reports a metaheuristic called Multiple Search Neuroevolution (MSN). A metaheuristic is a "strategy" that guides the search process, not an algorithm for a particular problem [94]. We report the MSN metaheuristic as a set of principles that shape an overall guide strategy. It tries to search multiple regions of the search space, while maintaining a certain distance between those regions to ensure diversity. Despite searching multiple regions simultaneously, it is a serial algorithm where populations (or samples as we will later call them) are generated and evaluated one at a time. Throughout this experiment the abbreviation MSN and the terms Our Algorithm & Metaheuristic will be used interchangeably, depending on context, to refer to the proposed metaheuristic.

Background and Related Work

A problem of Spaces

Training neural networks is essentially an iterative optimization process. The optimization parameters and objective function are the network's weights and loss function respectively. The optimization process aims to find a desirable set of weights given target input/output pairs such as in [96]–[98]. Through the weights, a mapping between the given input and the desired output is created. The weights are adjusted in order to approximate a function

that achieves said mapping. The larger or deeper the network, generally, the greater its capacity to approximate more complex functions. This is referred to as Representational Capacity in [6]. Increasing representational capacity allows the creation of more complex mappings, i.e. approximation of higher-order functions. In turn, the network can be used to address more challenging tasks.

In this context, it is important to examine the abstract concepts of Spaces. The search space is the space (set) of all possible weight configurations. The wider and deeper the network, the larger the search space. The type of neuron itself also affects search space. In a recurrent neural network, for instance, there is an additional recurrent parameter to learn. Besides architecture, another important factor is the range of the weights. That is, binary weights would lead to a considerably smaller search space than 32-bit representations (FP32). Weights in a neural network are independent. Each weight is a free-valued parameter, constrained only by the range of representation.

The observation space is the space (set) of all possible observations. Consider the case of the MNIST image dataset. The input size is a 28x28x1 full-precision (FP32) image (matrix). The entire dataset consists of 70,000 such images. The observation space, however, is made of all possible 28x28x1full-precision images (matrices). Thus the 70,000 images are not the entire observation space. They are simply a small fraction of it. The solution space can be defined as the space (set) of all possible outcomes of the network. If the network only has one binary output neuron, for example, then the entire solutions pace consists only of two members [0, 1]. Another example would be Generative Adversarial

Networks that generate images. The solution space is the set of all possible images (matrices). The larger the image to be generated, the larger the solution space, and generally the more difficult it is to find a high-quality solution.

Finally, there is loss space or gradient space. This is the space (set) of all possible error/cost values of the objective (loss) function. This is the space that the solver must traverse, in order to look for better solution candidate(s) in the search space. The topography of this space can sometimes be quite challenging to navigate. It may not be smooth, continuous, convex or even information-bearing. Salimans, Ho, Chen, Sido and Sutskever from OpenAI report this in [74] when they refer to cases where the gradients are "uninformative". Furthermore, the loss landscape can be riddled with local minima, wide valleys and other anomalies. Generally, the more challenging the loss space, the more difficult it is to improve upon a given solution.

The Search Process

Given all those spaces, it is perhaps now clearer how the search process is not simply a matter of architecture or size of the neural network. Each of the items mentioned above, e.g. network architecture, different spaces, etc. has a direct impact on the possibility and speed of attaining an acceptable solution. Towards that end, SGD uses information from first-order partial derivatives (gradients) of the loss function to aid it in the search process [99], i.e. gradient-following. Despite being computationally costly, gradients can be extremely informative. Over the decades since SGD was first invented, techniques such as Momentum [77] were introduced to improve the efficiency of the search process. In

addition, by using Backpropagation, and in turn the chain-rule, SGD can update an arbitrary number of weights in a neural network. It may sometimes suffer from complications e.g. vanishing and exploding gradients. There are of course techniques to tackle those, for instance residual connections [100].

Neuroevolution, on the other hand, does not use derivative information in its search process. Though computationally less-costly, not using derivatives can lead to the optimizer being inefficient and taking more optimization steps than SGD. Moreover, Neuroevolution requires the evaluation of a population of solutions (by performing the task and measuring performance) before taking a single optimization step. This makes it inherently penalized if one was to compare it directly with single-solution methods on number of evaluations. Without Backpropagation, Neuroevolution suffers when the trained network is relatively large. Each weight in the network becomes an additional dimension in the search space that can take any value within the representation bounds. Without clue or information on how to update each parameter, it essentially becomes a matter of educated guess.

Despite these apparent challenges, a simple genetic algorithm was able to solve Reinforcement Learning tasks using additive Gaussian noise, elitism and no crossover [75]. Note that a population size of 1000 was used. The evolved networks had 4M+ parameters, possibly amongst the largest networks ever evolved. Another result of that work was that Random Search performed surprisingly well, which may perhaps suggest something about the domain itself. In [74] smaller networks are also evolved for Reinforcement Learning

tasks, using Evolution Strategies [101]. Between 720-1440 workers, i.e. populations, were used.

Notably, Neuroevolution algorithms employed thus far are somewhat linear, as in [102]. There is no explicit notion of searching multiple regions in the search space. Instead, search would generally be concentrated on one region in hope to transition to more lucrative regions using the perturbation mechanisms. We sought to first address this aspect in the context of Global Optimization problems. The advantage of using Global Optimization functions is that the topology of the solution domain is known. It is possible to visualize optimizer behavior in the solution space, generate insights and determine how best to influence it. Following this line of thought, we describe the MSN metaheuristic in the next section.

Proposed Algorithm

A metaheuristic is a strategy that guides the search process [94]. It is not a singular algorithm for a singular problem. We introduce the MSN metaheuristic as a strategy to search multiple regions of the search space effectively. It consists of a set of smaller, locally-aware mechanisms and functions all operating to create a global aggregate behavior. Each of those mechanisms is described in this section, highlighting its function and import. In addition, many new terms and concepts are introduced and utilized. Equivalent terms, where relevant, are mentioned in order to maintain consistency with existing literature and norms. Since the mechanisms are numerous, they are divided into two groups. The primary group contains the core mechanisms without which the MSN

scheme can't function. The secondary group contains supplementary mechanisms that are introduced in effort to improve search efficiency under certain conditions.

Primary Mechanisms

1) *Pool Composition*: The sample pool is the set of candidate samples, i.e. populations. The pool size, i.e. number of samples, is always constant. It is extremely important to utilize the allocated pool as efficiently as possible. This is a major motivation in our introduction of MSN. In the case of MSN, samples in the pool are the neural networks' parameter vectors. They are initialized according to a weight initialization scheme, e.g. Xavier Normal [103]. After the first optimization step, i.e. generation, the pool will consist of the following components: **Elite**, **Anchors**, **Probes** and **Blends**. Let us introduce each of those in order.

   First, the Elite is the sample that collected the greatest reward since the optimization process began. Anchors are the highest-rewarded N samples in the current generation. They also need to be at least separated by a certain distance in the search space. This separation mechanism shall be introduced in a coming segment. From each anchor M probes are spawned. Probes begin as exact clones of their respective anchor after which each is randomly perturbed.

   It is possible that at any arbitrary generation the number of samples sufficiently apart is less than the allotted number of anchors, N. In such cases, the remaining

slots are filled with blends, i.e. crossovers, which will be described in a coming segment.

To summarize, the pool is composed of the **Elite**, N **Anchors**, M **Probes**, and **Blends**, should there are be any open slots. The size of the pool thus must be at least (N*M) + 1. The choice of the pool size enables, emphasizes or even disables the Blend mechanism.

2) *Perturbation and Adaptive Integrity*: Perturbation is the primary searching mechanism. It is the equivalent of genetic mutation, principally referring to the injection of noise into the makeup of the network. The magnitude and scope of that noise are determined according to (1**)** and **(2)**. Equation (1**)** describes the Search Radius, i.e. the magnitude of perturbation. We call it Search Radius because it can be thought of as defining the radius of a virtual circle around each anchor, within which probes will be cast in random directions. Generally, as the search radius increases, probes will be casted farther away from an anchor. In the same way that Search Radius defines the magnitude of perturbations, the scope of perturbations is defined by the Number of Selections in **(2)**. It determines how many of the weights shall be prone to noise, and how many will be preserved as is.

By examining (1**)** and **(2)**, it will be noted that they are functions of a variable called $\boldsymbol{integrity}$. It is a single real number in the range [0, 1], and the two equations are

designed to have attractive properties in that range. Thus, ***integrity*** governs the search radius and number of selections. It determines the exploration vs. exploitation aspect on a local scale. Each generation, the algorithm needs to decide on integrity. Increasing $integrity$ makes the search more exploitative and less exploratory, and vice versa. Perhaps determining the value of $integrity$ is the single most important decision the algorithm must take.

The value of $integrity$ is reduced, by a fixed amount, when the current generation of samples does not yield a score that is sufficiently better than the previous best score. A parameter in the algorithm defines the minimum accepted percentage of improvement in the generational top reward/score, we call it $Minimum\ Entropy$. For example, in the global optimization task, $Minimum\ Entropy$ is set to 1%. If the current generation does not improve upon the previous generation's reward by at least 1%, $integrity$ is reduced by a fixed step size. If it does improve by at least 1%then the current $integrity$ value is maintained. The $Search\ Radius$, a function of integrity, is calculated as

$$(1)\ \boldsymbol{SR(p) = \left(tanh\left((\lambda p) - 2.5\right) + 1\right) * lr}$$

where $p = (1 - integrity)$, λ and lr are scalar constants. The learning rate, lr, scales the function, controlling the bounds of the search radius. The shifted, scaled hyperbolic tangent function has attractive properties in the range [0, 1]. These properties are an almost-flat slope near 0 and an almost-flat slope near 1. This allows the algorithm to spend more time searching low-energy configurations even

as $integrity$ is reduced, where we estimate rewards are more likely. By flattening the slope near 1, it also prevents the algorithm from searching exceedingly high-energy configurations where it is unlikely to find rewards.

The $number\ of\ selections$ is calculated as

$$\textbf{(2)}\ \boldsymbol{Selections}\ (\boldsymbol{p}) = \frac{\boldsymbol{\alpha}}{\boldsymbol{1+\beta/p}}$$

where $p = (1 - integrity)$, and α and β are scalar constants. In the range [0, 1], this function starts at the origin and progressively becomes flatter as $integrity$ is reduced. This saturation limits the number of modifications in the network. Intuitively, making too many adjustments to a neural network in one step is usually unrewarding, especially when the configuration at hand is highly refined. It is unlikely that changing 70%, for instance, of a model's weights in one single generation will lead to a higher reward. This is especially the case when searching high-energy configurations, i.e. low $integrity$. The function is designed to saturate, in order to limit a situation where many cycles are wasted searching unprofitable regions.

To summarize, the perturbation mechanism is introduced with its two aspects called $search\ radius$ and $number\ of\ selections$. The value of $integrity$ controls perturbation in order to balance exploration and exploitation locally. Six hyperparameters are defined. Namely they are $step\ size$, $Minimum\ Entropy$, lr, λ, α and β. All values of those are determined empirically.

3) *Anchors, Minimum Distance and Probes*: Anchors are the N best-performing samples that are also separated in search space by a certain $Minimum\ Distance$. For example, if there are 5 anchors, those will be the 5 best-performing samples that meet the Minimum Distance criterion. Anchors are updated each generation.

The parameter $Minimum\ Distance$ defines the minimum separation requirement for a sample to become a candidate anchor. This assures searching different regions in search space. Let us walk through the anchor selection process. After sorting, the best-performing sample is picked as the first anchor. In an ordered reductive process, each sample is admitted as an anchor if it is separated from the other anchor(s) by at least $Minimum\ Distance$.

There are many possible distance metrics to choose from such as Euclidean distance. However, we wanted to use a metric that accounts for differences in both magnitude and position (of the weights), between the samples. Thus, Canberra distance was chosen, and it is calculated as

$$\textbf{(3)}\ \boldsymbol{d}(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{n}} \frac{|\boldsymbol{x_i}-\boldsymbol{y_i}|}{|\boldsymbol{x_i}|+|\boldsymbol{y_i}|}$$

where x, y represent the two parameter vectors, i.e. samples, under examination and i is each element in the vectors.

Finally, from each anchor M exact clones are created, called Probes. Each probe is perturbed, i.e. mutated, to produce a different version of the anchor. Thus, probes

search the local neighborhoods of the anchors. By having multiple anchors, multiple regions are searched in tandem. And by having multiple probes per anchor, multiple local neighborhoods are searched within those regions.

In summary, the processes of choosing anchors and probes are introduced. For anchors, the distance metric is chosen as Canberra distance, given by **(3)** to account for both magnitude and positional differences in the parameter vector. One hyperparameter is introduced called $Minimum\ Distance$. It defines the least acceptable distance between anchors, and its value is determined empirically.

4) *Blends*: Blending, called Crossover in other literature, mechanism combines randomly-chosen weights from two components (i.e. parameter vectors) to yield a blend (an offspring parameter vector). The first component is always one of the anchors, picked at random. The second component can be any sample in the pool. The first component is cloned to form the basis. A number of weights from the second component replace their counterparts in the basis. This number is calculated by **(2)**, introduced in the previous section. This is not dissimilar from human genomics where a child's genetic makeup is acquired from parent 1 and parent 2. Blending potentially allows the exploration of regions out-side the main mechanism of perturbation and its constraints. By being based on anchors, blends attempt to extend the actively searched area in the search space. This exploration behavior can be emphasized or discouraged by increasing or reducing the number of slots allotted to blends in the pool.

In addition, it is possible that not enough samples are sufficiently apart in the search space to fill the N spots allotted to anchors. This would also mean that not enough probes would be made, and thus many slots in the pool would remain unfilled. Blends fill all those slots, helping to reintroduce diversity into the pool. With diversity, distance between samples increases. In consequence, the allotted slots for anchors can be filled again.

In summary, this segment introduced blends and how they are picked. Blends attempt to explore regions beyond the constraints of perturbation. Also, they are important to fill open slots in the pool and maintain diversity. A visual overview of the system is given in Figure 14.

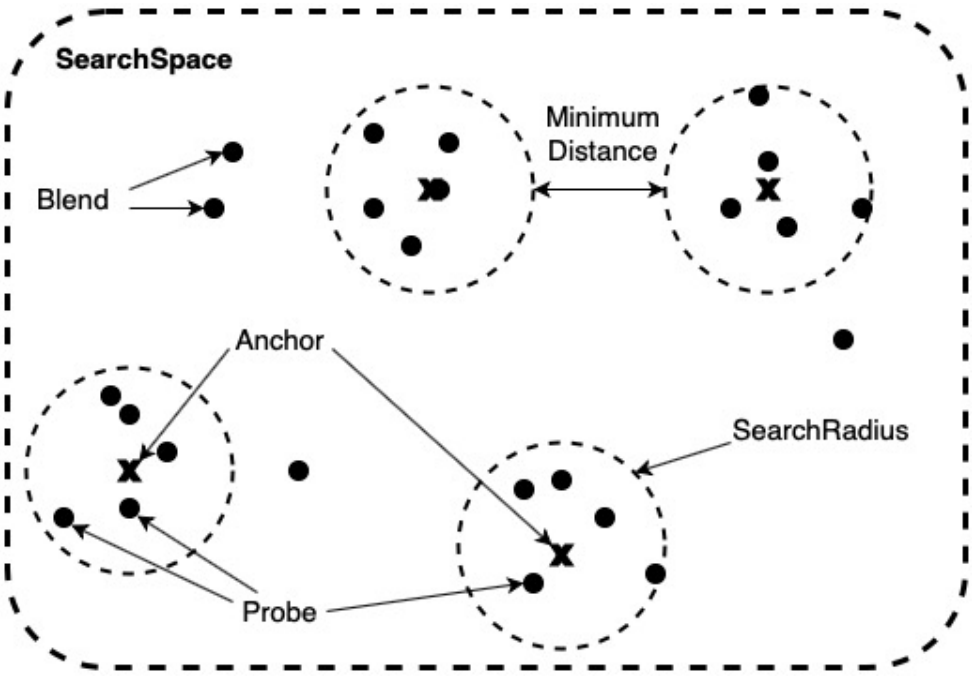


FIGURE 14 VISUAL OUTLINE OF PRIMARY SEARCH MECHANISMS OF MULTIPLE SEARCH NEUROEVOLUTION IN SEARCH SPACE

5) *Elitism*: This technique is well-known throughout; we mention it for completeness. There are two sorts of elites, a generational elite, and a historical elite. The generational elite is the best-performing, i.e. highest-rewarded, sample of the current generation. The historical elite is the best-performing sample across all generations.

   In MSN, the generational elite is always picked as the first of the Anchors. The historical elite is simply called the Elite. It is preserved as-is, without being subject to perturbation or blends, across generations unless another better-performing sample replaces it. The Elite is called upon whenever the mechanism of Backtracking is invoked, which will be presented in the coming segment.

SECONDARY MECHANISMS

6) *Backtracking*: Without enough improvement in the reward signal, $integrity$ is reduced. This makes perturbations and blends more potent, searching higher-energy configurations and encouraging exploration. This is a unidirectional behavior, and MSN will keep reducing $integrity$ (to its limit of 0) until an improvement is achieved. This has a couple of disadvantages, however. First, it may just be the case that the lucrative areas are in low-energy configurations. Due to the probabilistic nature of the algorithm, it is quite possible the algorithm "missed". Second, with low $integrity$ the samples become "hot" from applying high-magnitude perturbations on a large scope, and the weights start exploding.

Backtracking is a mechanism that resets $integrity$. It is triggered when MSN reduces $integrity$ for X consecutive generations, where X is a parameter called $Patience$. When it is triggered, backtracking resets the $integrity$ value back to maximum, and inserts the Elite as an Anchor in the pool. This accomplishes two things. First, by resetting $integrity$, the algorithm will search low-energy regions once more. Second, inserting the elite as an anchor, and spawning its probes with maximum $integrity$, helps "cool down" the entire pool. This is because the weights of the elite were not subject to perturbations. Thus, the search returns to low-energy configurations.

In summary, if $integrity$ is consistently decreased without improvement, the search process may need to be reset. Backtracking is mechanism to accomplish this. One hyperparameter is introduced, called $Patience$. It determines how many generations the algorithm should wait before backtracking, and its value is determined empirically.

7) *Radial Expansion*: Radial expansion increases the learning rate lr and α from (1) and **(2)** by a fixed percentage called $Expansion\ Factor$. Let us explain the reasoning behind this. Even at lowest integrity, there is a limit to how far a probe can be cast from its anchor, as defined by those equations. If that limit is too constrictive for the task, search efficiency will suffer for two reasons. The first: it can lead to the full number of anchors not being utilized, as the samples are not far

enough from each other. The second: it can increase the number of exploratory search iterations since the algorithm is too conservative. Radial expansion is triggered whenever the number of anchors becomes less than the allotted number, N.

In summary, the mechanism of Radial Expansion decreases the constrictions of local search. It is an attempt to utilize the complete number of anchors. One hyperparameter is introduced, called $Expansion\ Factor$, and its value is determined empirically.

Implementation

For all experiments in this experimental study, a pool size of 50 samples is used. This is a remarkably low number, compared to other works using evolutionary algorithms on neural networks such as [75] and [74] using more samples by one or two orders of magnitude. This choice, we believe, emphasizes efficiency. Thus, with a relatively small pool, the goal is to converge in the upcoming tests in as little iterations as possible. The algorithm is implemented using the PyTorch framework [104]. The information contained in this study should be enough for any developer to implement the algorithm.

The training process with MSN can be described as follows. Since MSN is population-based, a pool of networks is being evolved simultaneously. At the beginning, each network is initialized according to a weight initialization scheme, we use Xavier Normal. In each iteration, i.e. optimization step, a query of the environment is conducted by each network.

The networks each take the same input and produce their own output. Then the loss/reward/cost signal is computed and fed to the algorithm. It informs the algorithm about the quality of each network's output. Finally, MSN adjusts the weights of the networks and the process is repeated.

The experiments run on an Nvidia DG-X desktop computer, featuring four Nvidia Titan V GPUs. In the Global Optimization task, the experiments run on CPU and a single GPU. In the MNIST hand-written digit classification task, the experiments run on CPU and the four GPUs for data parallelization. The models are copied into each GPU and inference is conducted by dividing the training set/batch into four chunks, computing loss and aggregating the result. This is only to speed wall-clock time and does not affect the performance of the algorithm or the results of inference. For all experiments a pool size of 50, with 4 anchors, and 8 probes per anchor are used.

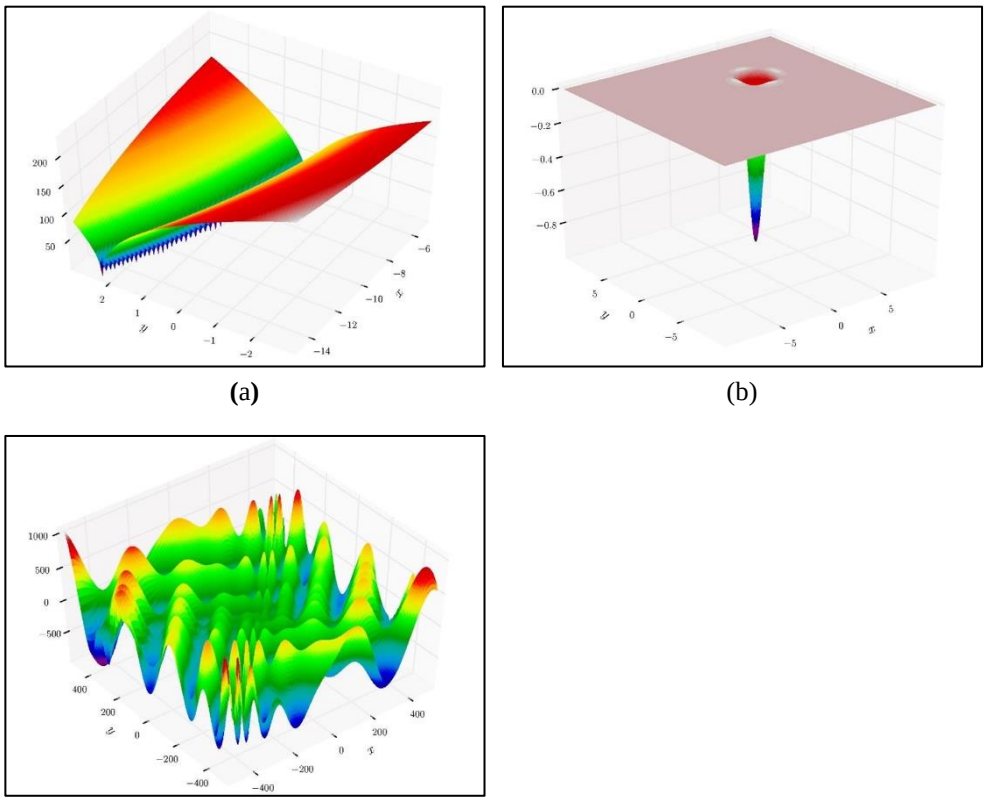
Task 1: Global Optimization Functions

We test MSN on two sets of Global Optimization functions. The first is a standard group and part of the BBOB challenge suite [105], a measure introduced in 2016. Those functions are also commonly implemented in different Python libraries, and thus allow us to compare MSN to other evolutionary algorithms. The second set is a special group of functions that are less common, and neither a part of the BBOB nor implemented in evolutionary libraries. Thus, a comparison with other algorithms on that set was not possible. Nonetheless, the challenges they pose are unique due to their irregular topographies. We thought it could be helpful to measure the performance of the algorithm against them.

The optimization task is straightforward and simple. Starting from a random location (x, y) in the search space, the algorithm needs to find the global optimum of the 2-dimensional function or approximate it. If the algorithm is within 0.06 of the global optimum value, the search terminates as a success. For example, if the global optimum is 0, then as soon as the algorithm reaches 0.06 or less it terminates. The number of optimization steps taken until termination is recorded and it is the measure of performance in the experiments. The lower the number of optimization steps needed to converge, the better the algorithm performs. The algorithms also terminate automatically if the number of steps taken exceed 5000. It is unlikely that the algorithm would converge beyond that point, and for reasons of comparison, it would not mean much if it did either.

In Task 1, MSN optimizes a neural network model with a single hidden-layer of size 128. The network takes two real numbers (x, y) as the origin, picked uniformly from the observation space and remain constant throughout that experiment. The uniform distribution's limits differ for each function. The network outputs two real numbers, its prediction of the coordinates of the global optimum. Other evolutionary algorithms don't optimize neural networks, they operate directly on the problem. The implementations of those algorithms are found in the Inspyre and PyBrain evolutionary libraries [106], [107]. The default parameters are used. Generally, parameters in standard library implementations are either directly copied from the algorithm's paper or picked to suite a wide set of problems. They are certainly not adversarial. For a fair comparison, however, we did not tune our algorithm, MSN, parameters either. We determine suitable values, empirically, and then use that exact same set across all experiments.

For every function, the optimization experiment is repeated **five times per algorithm**. The average number of optimization steps, i.e. generations, until termination is recorded. Limits for the functions are given as a single pair for symmetrical limits, and in the form (x-start, x-limit, y-start, y-limit) for asymmetrical limits. They are: [-5, 5] for Ackley, [-5.2, 5.2] for Rastrigin, [-2, 2] for Rosenbrock, [-500, 500] for Schwefel, [-15, -5, -3, 3] for Bukin N. 6, [-20, 20] for Easom and [-512, 512] for Eggholder. The same limits are used in all experiments. The definitions of the functions can be found in [108]. The first group of consisted of the Ackley, Rastrigin, Rosenbrock and Schwefel optimization functions. The tested algorithms are Evolution Strategies (ES) [101], Particle Swarm Optimization (PSO) [109], Differential Evolution (DE) [110], Simulated Annealing [111], Fitness-Maximization Expectation (FEM) [112], Policy Gradients with Parameter Exploration (PGPE) [113] and Random Search. For some of those algorithms, not all the functions were available in their libraries for testing. However, at least two functions were tested in each case.



(a) (b)

(c)

FIGURE 15 3D PLOTS OF THE SPECIAL GROUP OF GLOBAL OPTIMIZATION FUNCTIONS. CREDIT [114]–[116]

The mode of comparison for the algorithms is the number of optimization steps performed. All of them had a population/swarm of size 50. It remained to be seen how well each will use the available resources to guide the optimization process.

The special group of functions is composed of the Bukin N. 6, Easom and Eggholder functions. Let us examine each in turn. The Bukin N. 6 function has a unique feature of an extremely narrow valley where the global optimum lies. This poses a challenge for the exploitation aspect of an optimization algorithm. If an algorithm takes too large optimization steps, it is unlikely to find that lucrative strip. The Easom function is practically flat everywhere except a thin spike, where the global optimum lies. The

algorithm needs to explore as fast as possible the search space, and then once it finds the spike, to being exploitation and travel towards the global optimum. It showcases a balance between strong exploration and exploitation. Finally, the Eggholder function is extremely irregular and adversarial in its shape. Its global optimum is literally in a corner. The optimizing algorithm needs to overcome the large number of deep local minima and maintain the search for the global optimum. Of all the functions, it has the largest solution space. A visual representation of the functions' landscape is given in Figure 15.

TASK 2: IMAGE CLASSIFICATION

For Task 2, a convolutional neural network (CNN) is trained to solve the MNIST hand-written digit classification problem. It is a standard entry-level problem where the goal is to correctly classify 28x28 greyscale images according to the number they feature. The CNN model consists of four convolutional layers of size 32, 64, 128 and 128 respectively with stride 2 and max-pooling in between, followed by a single fully-connected layer of size 512. Parametric ReLU non-linearity in PyTorch was used. The model has 4.7M parameters.

The following set of conditions were imposed during the experiments. Since MSN is population-based, a pool of networks is being evolved simultaneously. This stretches inference time linearly. To speed up computations, and leveraging NVIDIA GPU Tensor Cores, half-precision floats (FP16) are used. Furthermore, only a subset of 2000 randomly-picked images (3.33%) of the MNIST training set is used. However, the entire validation set is used. We set the termination condition to be a loss of 0.15. Given only a subset of the

training set, demanding further improvement is likely to introduce a cycle of diminishing returns. All this comes at a cost of not reaching the best possible performance on the task, but that is not the concern of this work. The goal from this task is to assess the suitability of MSN to optimize relatively large neural networks with 10^6 parameters.

As a baseline, the same CNN model is trained under the same conditions with SGD. Note that SGD uses mini-batch training. It updates the weights after inference on every mini-batch. By comparison, MSN does not use mini-batch training. It performs a weight update after inference on the entire training set (2000 images). For that reason, comparing SGD to MSN on the number of absolute inferences would be inherently flawed. Moreover, recall that these two belong to different families of optimization algorithms, one is a single solution while the other is population-based. The experiment is repeated five times and the mean is reported.

Results

Task 1: Global Optimization Functions

TABLE 12 EXPERIMENTAL RESULTS OF OPTIMIZATION ALGORITHMS ON TYPICAL GLOBAL OPTIMIZATION FUNCTIONS. A FEW ENTRIES ARE MISSING BECAUSE THE CORRESPONDING LIBRARY IMPLEMENTATIONS ARE UNAVAILABLE. THE SPEEDUP COLUMN SHOWS THE IMPROVEMENT IN OPTIMIZATION STEPS WHEN USING MSN METAHEURISTIC AGAINST ALL OTHER ALGORITHMS, DISCOUNTING THOSE THAT DID NOT CONVERGE.

Function	**Number of optimization steps**								
	MSN	ES	PSO	DE	SA	FEM	PGPE	RS	Speedup
Ackley	**17**	36	117	659	5000+	287	152	4149	2.1-244X
Rastrigin	**49**	2020	418	632	2368	2389	421	3074	8.5-63X
Rosenbrock	**20**	67	730	2415	2398	-	-	-	3.3-120X
Schwefel	**113**	2019	492	2310	5000+	-	-	-	4.3-20X

Table 12 presents the results of the experiments from the first group. The results warrant some analysis. Some methods fared well on some functions but struggled on others, such as Evolution Strategies. Some methods struggled consistently, such as Simulated Annealing which failed to converge on two occasions. Unlike the findings in [75], Random Search performed poorly. As expected, it may be a subject of the extremely limited pool size we use, as well as the problem domain. The strongest competitors were Evolution Strategies, PSO and PGPE. The consistent best-performer, however, was MSN.

The improvement to using MSN compared to the other algorithms is calculated and presented in the Speedup column. At least, MSN reduced the optimization steps by 2X, and at best by 244X. As expected, the Schwefel function took the longest for MSN to solve.

TABLE 13 EXPERIMENTAL RESULTS OF RUNNING MSN METAHEURISTIC ON OPTIMIZATION FUNCTIONS WITH SPECIAL PROPERTIES.

Function	Number of optimization steps
Bukin N. 6	28
Easom	9
Eggholder	170

The results of the experiments on the group of special optimization functions is given in Table 13. In all cases, MSN converged in a relatively low number of iterations. Particularly in the case of the Easom function. By searching multiple regions separated by a distance metric, the algorithm is naturally exploration-oriented. The task of traversing a smooth-but-flat surface did not strain the algorithm. Similarly, by using the Probe mechanism, it is also naturally exploitative. Thus, finding and exploiting the extremely narrow valley in the Bukin N. 6 function was not taxing. The most challenging function was, as expected, Eggholder. Its highly irregular landscape and having its global optimum in a corner proved challenging. Recall also that its solution space is the largest.

TASK 2: IMAGE CLASSIFICATION

TABLE 14 EXPERIMENTAL RESULTS OF RUNNING OPTIMIZATION ALGORITHMS ON MNIST DATASET, USING 3.33% OF ITS TRAINING DATA.

Algorithm	MSN	SGD	Speedup
Number of optimization steps	2333	320	-7.3X
Validation Accuracy (%)	90	90	-

From Table 14, MSN takes 2333 generations to converge to the target training loss. This corresponds to 90% validation accuracy. In general, MSN can train the relatively large network, using a pool of only 50 samples. Compared to Task 1, the observation space is

larger by two orders of magnitude. The search space also is larger by four orders of magnitude.

Despite this, MSN takes only an order of magnitude more steps to converge for MNIST than the Eggholder function.

On the other hand, SGD can reach the target training loss taking only 320 steps. The speedup factor is -7X, since instead of being faster, our algorithm is slower than the baseline. That MSN would be slower than SGD is not surprising. Recall that SGD utilizes first-order partial derivatives (gradients) to guide its search process. On the other hand, MSN does not have access to such information, being derivative-free. Remarkably, however, it is still within an order of magnitude of baseline.

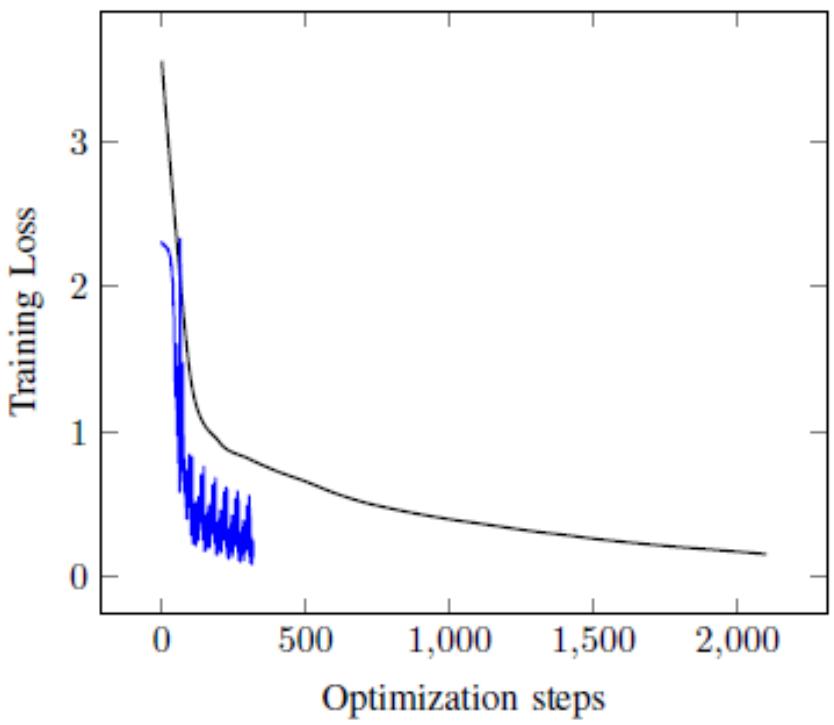


FIGURE 16 TRAINING PROGRESS OVER OPTIMIZATION STEPS FROM A SAMPLE OF THE EXPERIMENTS PERFORMED ON TASK 2. EXPERIMENTS TERMINATE ONCE TRAINING LOSS OF 0.1 IS ACHIEVED. AS THE TRAINING PROGRESSES, THE GAP BETWEEN MSN AND SGD NOTABLY WIDENS. THE BLACK TRACE IS MSN AND BLUE IS SGD.

Figure 16 shows sample of training progress over optimization steps. It is drawn using experimental data from one of the five trials performed. In that experiment, the target training loss was achieved at step 320 for SGD, shown in Blue, and 2161 for MSN, shown in Black. Our algorithm was 6.75X slower than baseline in that case. The figure also showcases how swiftly training loss decreases in the early stages, for both algorithms. However, after going below a training loss of 1, SGD continued to improve at a slightly slower rate while MSN started to plateau. This perhaps indicates a limitation in the exploitative nature of MSN for large search spaces.

The figure also suggests that SGD has not fully converged. If allowed more steps, it would have likely reduced the loss further. For MSN, the algorithm seemingly started to plateau at step 1,500 and every further step achieved diminished returns.

As such, the gap between MSN and SGD widens as training loss is reduced.

Using only 50 populations, the CNN trained in Task 2 is by far the largest trained using derivative-free methods. While not state-of-the-art, the performance of the network meets the predefined target. To the best of our knowledge no other work managed to train relatively large neural networks under such constrained conditions. In comparison to recent publications [75] and [74] that attempt to use derivative-free methods to train neural networks, we use fewer populations by 14-28X. The task domains were different, however.

Conclusion

Drawing upon the limitations of derivative-based single solution optimization methods, this experimental study introduced a new metaheuristic called Multiple Search Neuroevolution (MSN). It is a derivative-free population-based strategy that guides the optimization process of deep neural networks. Its ensemble of mechanisms is presented in detail, divided into two groups. An implementation of the MSN metaheuristic is tested on two tasks. In both, MSN optimizes a pool of neural networks to solve the task. The first task was to find the global optimum for groups of common and special global optimization functions. In solving the problems, MSN reduced the number of optimization steps by 2-244 X, compared to baseline evolutionary algorithms. Featuring nine empirically-derived

hyperparameters, however, MSN is certainly not as simple as the baseline. Thus, there is a clear trade-off between speedup and complexity.

The second task was to reach the target training loss on the MNIST hand-written digit classification dataset. Using 3.33% of the training set, MSN required 7X more optimization steps then the baseline algorithm (SGD) to reach the termination condition. This was anticipated since SGD utilizes gradient information to guide the search process, while MSN is derivative-free. The results further suggest a limitation in the exploitative mechanisms. No doubt under such a constrained pool size, it is challenging to balance exploration and exploitation. However, perhaps more attention is due towards the exploitative aspect.

The study has some limitations. First, only two-dimensional functions were used. In future work, testing higher dimensional functions would be helpful. Second, only algorithms available in standard evolutionary libraries were used as baseline. In future implementations, it would be helpful to compare MSN to more competitive evolutionary algorithms. Finally, since MSN is presented as a collective, and the role of each mechanism separately is not quantifiable. Performing an ablation study would be helpful to investigate the individual effect of each search mechanism, for different types of problems.

Nonetheless, the results show that there is credence to the mechanisms introduced. Consistently outperforming the other evolutionary algorithms in Task 1 is a significant indicator. The model in Task 2 featured 4.7M parameters, which is a significant search space for evolutionary methods. In addition, only 50 populations were used. Yet, MSN was able to train to 90% validation accuracy using 3.33% of the available training set.

CONCLUSION

Multi-candidate derivative-free optimization methods were used in attempt to overcome the limitations of Deep Learning, i.e. single-candidate derivative-based optimization methods. This decision follows from the previous chapter where Deep Learning was found to be fundamentally limited in some respects, e.g. exploration. First, we investigated simple evolutionary algorithms in the form of the Batchu algorithm on simple problems. The chosen simple task at first was the flappy bird video game. By developing Accelerated Neuroevolution, we substantially improved upon the performance of the Batch algorithm in that task.

Accelerated Neuroevolution was then used to test ERL on a real robot setting. The robot learned to perform the correct head movements in the object-centering task experientially. The experiment was a successful demonstration of ERL since the robot achieved a score of 7 on our AS scale.

The AN algorithm has limited mechanisms to ensure diversity, however. It requires manual tuning of some parameters, however. If an algorithm was going to be used to make robots learn experientially, it first needs to demonstrate strong effectiveness to solve high-dimensional problems using neural networks.

For those reasons the Multiple Search Neuroevolution algorithm was developed. The goal was to overcome some of the limitations on AN. Tests on Global Optimization functions revealed the strengths of MSN. It performed remarkably well on complex functions using

a limited number of populations. This revealed how well the mechanisms in MSN are performing.

The MNIST task also revealed the weaknesses of MSN. Namely that it is slow to converge and requires high sample-complexity. In scenarios where obtaining a sample is costly, i.e. real robots in the physical environment, these weaknesses are critical. We do not see this as a shortcoming of just MSN, but of multi-candidate methods in general. For this reason, it is important to investigate and improve upon single-candidate methods.

CHAPTER 5: TRAINING DNNS WITH LOCAL SEARCH

INTRODUCTION

Following the experiments of the previous chapter, we discovered that multi-candidate methods are not feasible in general for robot problems due to their inherent high sample-complexity. In this chapter, we investigate single-candidate methods in order to improve upon sample-complexity. It is imperative to test any new algorithm on benchmark problems first, before physical experiments.

EXPERIMENTAL STUDY 4: TRAINING NEURAL NETWORKS USING LOCAL SEARCH

INTRODUCTION

Deep neural networks have found many applications over the past few years. From Computer Vision [81] to Robotics [88], neural networks were used by experts and successfully demonstrated their wonderful potential. Deep models perhaps earned their attractiveness from their immense representation capacity and the automatic learning of features from data [6]. Stemming from such undeniable interest, this study aims to introduce an optimization algorithm to train neural networks. The goal is to simplify the training process, and perhaps complement existing optimization algorithms by offering yet- another-tool for the researcher or practitioner.

This experimental study focuses on the use of the Local Search (LS) algorithm to train neural networks. LS is a single-candidate derivative-free optimization algorithm. It differs

from evolutionary algorithms, e.g. [101], which are also derivative-free but multi-candidate. The Local Search algorithm can be useful in cases where the neural network or task is non-differentiable, does not have a representative loss function or the derivatives are uninformative in guiding the optimizer [74]. The study focuses only on the Multi-Layer Perceptron (MLP) neural implementation, i.e. feed-forward networks. The Local Search algorithm, however, may be useful in other neural network implementations. In the following paragraphs, the training loop of neural networks will be examined.

Typically, training is an iterative optimization process before deployment. Its goal is to find a set of parameters (weights and biases) suitable for the goal task. The network's parameters are modified by an optimization algorithm in a loop until convergence. When put together, the network's parameters can be thought of as a single vector $\boldsymbol{\theta} \in \mathbb{R}^N$, where N is the space dimensionality and is possibly on the order of 10^6. The optimizer moves the parameter vector $\boldsymbol{\theta}_i$, in parameter space (set of all possible values), where i is the i-th iteration in the optimization process.

Think of $\boldsymbol{\theta}_i$ as a point on a hypersurface of N orthogonal dimensions. In such a scenario, knowing the direction in which to move the solution (which elements in θ to increase, and which to decrease) is extremely valuable. Computing the gradient of the loss function usually tells the optimizer about the direction (in parameter space) in which it should move the current solution $\boldsymbol{\theta}_i$ to produce the candidate solution $\boldsymbol{\theta}_{i+1}$. This information about search directions has made gradient- based optimization algorithms a staple in neural net training.

The typical training process exposes the neural network to samples from input space, and evaluates the outputs of the networks through what is called a loss function. By minimizing training loss, which is a function of network's parameters, the neural network learns important features in the input space. Those features usually generalize beyond the training dataset, and are validated using the validation dataset. This is the typical training dynamic in Deep Learning. The choice of loss function is thus quite important. In well-defined tasks such as Image Classification [100], there are widely-used loss functions such as Categorical Cross-Entropy. For other ill-defined tasks, a representative loss function may need to be "designed" by an expert.

The proposed algorithm (LS) does not need a loss function to train the network. It only needs a "training signal" that may be from a loss function or may indeed be from another source. In this, LS makes training simpler. However, to be sure, having a loss function may still be useful in many cases. In addition, LS uses randomness to search local spaces instead of the global parameter space. As such, on a high level, the philosophy is to turn the vast global space into a smaller local neighborhood. The search takes place in only a constrained number of directions concurrently. This will be discussed in the coming sections.

Training Accessibility

In a well-defined setting, it is common to find representative loss functions that can be used to tackle the task at hand. An example of this is Image Classification. It is a well-defined task, and usually loss functions such as Categorical Cross-Entropy can be used to train networks to solve it. However, there are other sorts of settings where loss functions may

not be available or representative. An example of this is Hyper-parameter Tuning. There is no loss function that an optimizer can use to solve this problem. Another example is Reinforcement Learning. While the Bellman equation is commonly used, it is not always representative of the task. An expert is usually needed in this case to design a loss function for the task.

The Local Search algorithm can be used by non-experts as an alternative or as a complement to traditional optimization algorithms for neural nets. The LS algorithm does not require a loss function to train a neural net. The simplicity of the algorithm, the simplicity of its implementation and the fact that it does not require a loss function can make it ideal for non-experts or those who want to try neural networks for the first time or on novel tasks. In a nutshell, it can make training neural networks more accessible to more people.

Related Work

Despite creative variations, e.g. GANs [96], fundamentals of neural network training have remained relatively constant with the reliance on the Backpropagation algorithm [78] and Stochastic Gradient Descent [15] variants such as ADAM [17]. This stagnation in innovating neural network training methods has also been remarked upon in [117] and [118].

The authors in [118] proposed a new training approach called Decoupled Neural Interfaces (DNI). This method is an alternative to Backpropagation. It is still derivative-based, as it uses gradient descent. They validated their approach on a number of tasks including the

MNIST and CIFAR-10 datasets. The method performed competitively in terms of final performance, compared to BP. The experiments were performed on relatively long optimization horizons (500K steps). Compared to DNI, BP was extremely quick to converge. However, DNI was primarily developed to address the “Locking” problem in the BP algorithm, and it is successful in that regard.

The Kick-back approach proposed in [117] is also presented as an alternative to backpropagation. Like DNI, it is still derivative-based. The approach is validated on simple robotic datasets, and not complex ones such as MNIST or CIFAR.

The authors remark that their method was still in its infancy, and thus not suitable for modern deep learning tasks such as multiclass learning (e.g. Image Classification). Kick-back was mainly addressing the credit-assignment problem, however. The performance of Kickback was similar to BP in terms of convergence speed and accuracy.

In recent years there were attempts to propose alternative neural network training methods in different settings. For example, researchers in [75] and [74] used Genetic Algorithms and Evolution Strategies to train a network to solve typical reinforcement learning tasks such as Atari games. Both algorithms performed well on the given tasks. Both are evolutionary algorithms and use a population size on the order of hundreds to a thousand. At each optimization step, hundreds of function evaluations need to take place before the optimizer can produce the new set of candidate solutions. Thus, this approach can be costly in terms of number of function evaluations, especially when compared to single-candidate optimization approaches.

There is little evidence to suggest a focus in the machine learning community on variants of Random Search, such as our Local Search, as means to train neural networks. There were attempts in the 1990s such as [119] and [120] to train neural nets with Randomness-based algorithms. The tasks in those papers, however, are far less complex than modern deep learning tasks. The networks themselves were much smaller and did not contain millions of parameters as is common nowadays. Interestingly, it was remarked upon in [75] that random search performs exceedingly well to train the neural network, and does in fact solve Atari games. Random Search has been found to produce even "high scores" in those games, beating modern deep reinforcement learning methods such as DQN [121].

Algorithm

Local Search

Local Search is built around the concept of managing the immense dimensionality of neural network parameter space. The main approach is to reduce the global search dimensions, N, into a local neighborhood composed of a relatively small number of dimensions, S. Instead of searching N dimensions simultaneously, the LS optimizer partitions the space into $K = N * S$ batches. Every iteration(s) it will search a segment for a number of subsequent iterations. To move in all N dimensions, K iterations need to elapse. The dimensions in each batch are picked at random by shuffling an indices vector $I = [0, 1, 2, \ldots, N - 1]$ and then splitting it into K batches of size S. The shuffling is repeated every K iterations.

The intuition in this approach is as follows. When searching a dimension, there is no information to guide the optimizer on which direction (positive or negative) to move along,

and by how much. This leaves the matter to a pseudorandom guess sampled from a uniform distribution $\mathcal{U}(-r, r)$, where r is an empirically-derived hyper-parameters. In aggregate, the guesses are defined as a noise vector **d** whose components are independently sampled from $\mathcal{U}(-r, r)$. The more concurrent "guesses" the optimizer has to take, i.e. as S increases, the less likely it will yield a positive outcome. Thus, randomly searching all dimensions at once is extremely unlikely to yield improvements in the score. Consequently, the idea is to avoid the pitfalls of a large-dimensional random search by only searching a quite small number of dimensions, i.e. perform a local search. Naturally, doing this will come at a cost of convergence speed.

Score Decay

During training, the optimizer can get stuck in a local minimum. By being "greedy", the optimizer can discard solutions that perform slightly worse than the current solution but would lead to a better solution over the proceeding optimization steps. Score Decay worsens the score (i.e. increases if the objective was to minimize, or decreases if the objective was to maximize) decreases by a certain amount p_{score} (0.0002 in this study) each step. It is designed to help the optimizer move out of local minima and explore the search space more actively. The value of p_{score} is determined empirically. It should not be too large, else the optimizer may diverge.

It should be noted that using Score Decay may cause the score to oscillate between better and worse values. In other words, without Score Decay, the score should monotonically improve or remain constant but never worsen. In the ablation study that will follow, the

importance of Score Decay will be examined. The entire algorithm is given in Algorithm 1.

ALGORITHM 1 LOCAL SEARCH W/ SCORE DECAY (MINIMIZATION MODE)

Result: $\boldsymbol{\theta}$
Initialize model: $\boldsymbol{\theta}_0$;
Declare noise vector of size S: $\boldsymbol{d} \in [-r, r]^S$;
Initialize indices vector: $\boldsymbol{I} = [0, 1, 2, 3, \ldots, N - 1]$;
Prepare I( )
    Shuffle $\boldsymbol{I}$;
    Split $\boldsymbol{I}$ into K batches of size S;
    Declare batch index: $k = 0$;
Declare penalty: $p = p_{score}$;
$score \leftarrow F(\boldsymbol{\theta}_0)$;
for $i \leftarrow 0$ *to* 200,000 **do**
    if $k > K$ **then**
        Prepare I( )
    $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i$;
    Sample noise from Uniform distribution: $\boldsymbol{d} \sim \mathcal{U}([-r, r]^S)$;
    Add noise to selected indices in $\boldsymbol{\theta}_{i+1}$: $\boldsymbol{\theta}_{i+1}[\boldsymbol{I}_k] = \boldsymbol{\theta}_{i+1}[\boldsymbol{I}_k] + \boldsymbol{d}$;
    if $F(\boldsymbol{\theta}_{i+1}) < score$ **then**
        Update rules:
        $score \leftarrow F(\boldsymbol{\theta}_{i+1})$;
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{i+1}$;
    if *convergence OR stagnation* **then**
        Terminate;
    Score Decay: $score = score + p$;
    Increment batch index: $k = k + 1$

EXPERIMENTATION

This section presents the experiments that were performed using the Local Search algorithm. LS is tested on the Fashion-MNIST dataset. The choice of this dataset is not arbitrary. In many papers, it is standard procedure when introducing a new concept to first test it on the hand-written digit classification dataset MNIST. However, over the years, it

has been remarked that this dataset is not quite representative of modern ma- chine learning tasks, particularly in Computer Vision. For that reason, the machine learning community began introducing different datasets, and FashionMNIST is one of those. The classes are much more challenging than regular MNIST.

The trained model has 5+ Million parameters. The model is a VGG-like CNN composed of: 2 convolutional layers with 32 and 64 kernels, a maxpool layer, 2 convolutional layers of 128 and 256 kernels, a maxpool layer, a fully-connected layer with 6400 neurons, a fully-connected layer with 768 neurons and an output fully-connected layer with 10 neurons representing the 10 classification classes. In this work, the activation function used after each layer is the hyperbolic tangent function. However, the Rectified Linear Unit function (ReLU) has also been tested and was found to yield similar results.

There are 4 experimental sets. The first is an overview of optimization algorithms' performance on FMNIST. The second is a minor ablation study on the LS algorithm. The third is a presentation of hyper-parameter tuning. Finally, the fourth is a training of the CNN with the Local Search algorithm using only the training accuracy as a learning signal, i.e. not using a loss function as is typical in SGD-based optimization.

The first experimental set is a direct comparison on training performance between Stochastic Gradient Descent (SGD+BP), Local Search (LS) and Random Search (RS). Each algorithm is allowed to converge fully. If it is clear that the algorithm has converged or will not improve further, it is terminated. The aim is to showcase Local Search between an upper-bound (SGD) and a lower-bound (RS).

The second experimental set is a minor ablation study on two features of the LS algorithm. In each case, the same conditions are used except for the specific feature under examination. The most obvious feature is Locality. Thus, the first experiment is to conduct Global Search (i.e. use all directions). The second experiment is to remove the Score Decay feature. If the algorithm does not converge or it is clear it will not improve, training is terminated.

The third experimental set is an examination of the founding hypothesis of using Local Search to train neural nets: that the greater the number of dimensions the search is conducted over, the lower the likelihood of improvement is. In this set, the number of search dimensions is varied.

The fourth experimental set demonstrates the usefulness of LS when there is no loss function. Without a loss function, derivative-based methods would not be usable. In such a scenario, only derivative-free optimization methods are usable. The experiment examines the possibility of learning using only the training accuracy as a learning signal. The algorithm used to train the network is still Local Search. However, instead of minimizing the loss value, the algorithm attempts to maximize training accuracy. It should be noted that switching from loss to accuracy will likely require a re-tuning of LS hyper-parameters, such as S and r to achieve best performance in this task. We did not do re-tune the hyper-parameters, however, for the sake of conformity. The results of the four experimental sets are presented in Figure 17, Figure 18, Figure 19 and Figure 20, as well as their corresponding tables Table 15,

Table 16, Table 17 and Table 18.

Training Constraints

Local Search requires a static loss surface and thus is not amenable to mini-batch training as is used in SGD, which can estimate the gradient from a mini-batch. We can't use mini-batch training, but we nonetheless have three options. The first is to download the entire training set to GPU memory, and run inference on it. However, the FMNIST has 50,000 training images and could not be loaded entirely onto the GPU (Titan V 12GB).

The second option is to divide the training set into mini-batches in RAM, send each batch for inference on the GPU and then aggregate the metrics (loss and accuracy). This option means that the entire training set can be used. It comes at a cost of wall-clock time, however. There are more than 10 experiments performed in total in this study. Each experiment runs until the algorithm terminates, which can take up to hundreds of thousands of iterations. Running inference on the entire training set for millions of iterations is prohibitively time and resource-consuming.

The third option and the one used in this study is to download only a portion of the training set, as much as will saturate the GPU memory without blocking it. Inference is much faster, and the conclusions and insights drawn from the experiments should still hold. Thus, in this study, only 10% of the training set is used (5,000 images), picked at random.

In the case of validation, the entire set is used. Validation is performed only after the algorithm converges or is terminated, i.e. it is only performed once. For the validation phase, it is not prohibitive to divide the validation set into batches, send each to the GPU

and aggregate the metrics (i.e. follow option 2 mentioned above). As such, in the validation runs, the entire validation set is used (10,000 images).

The relatively long horizons (high number of optimization steps) in this study obviate the need to generate confidence intervals. If there were a few "lucky" steps in the optimization trajectory, they would not affect the global behavior of the algorithm. Thus, the experiment does not need to be repeated many times to assert performance.

RESULTS

Each experiment will be discussed in turn. The first experimental set's results, as shown in Figure 17, reveal that LS is not as fast as SGD in terms of convergence speed. It approximately takes SGD 1,000 steps to converge to a loss $\simeq 0.15$. LS takes 50,000 steps to reach the same loss, i.e. LS is up to 50X slower. This is not the whole story, as it is noticeable from the figure that the gap between SGD and LS widens as the optimization advances. It means that the 50X margin is a worst-case for this particular loss hypersurface. Interestingly, however, LS continues to improve until it reaches a loss of 0. SGD simply stagnates at 0.15. It may be suggestive of a resilience property of Local Search compared to Gradient-following approach.

TABLE 15 RESULTS OF EXPERIMENTAL SET 1

Name	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
LS	0.0165	100	0.6760	84
SGD	0.1431	97	0.4024	86
RS	2.7617	9	2.7552	9

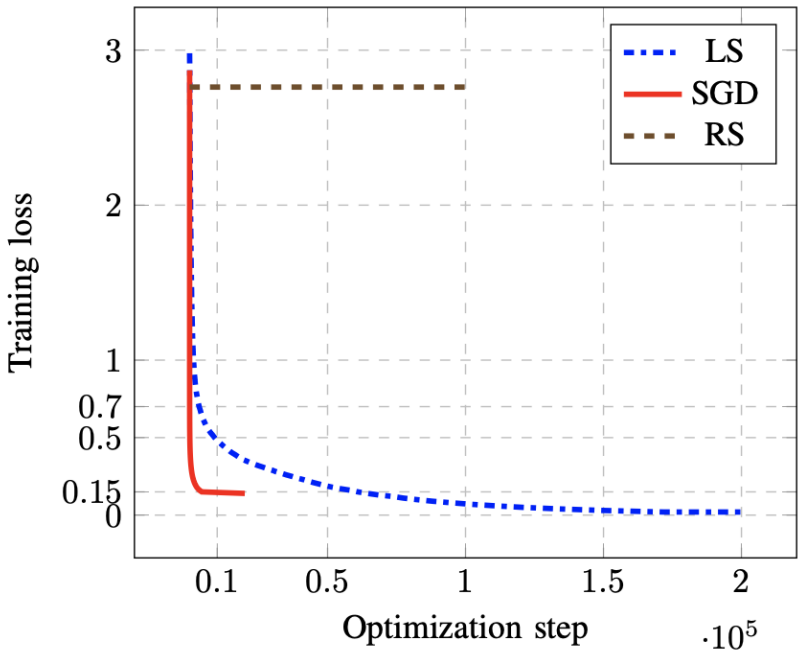


FIGURE 17 LOCAL SEARCH TAKES ~ 50X MORE STEPS THAN SGD TO REACH THE SAME LOSS. HOWEVER, LS CONVERGES TO A 0 LOSS, BEATING SGD. RANDOM SEARCH UTTERLY FAILS.

In addition, it is clear that Random Search fails completely in this scenario. It does not improve the solution at all, even after running for 100,000 steps. This comes in contrast to earlier findings in [75]. In that paper, RS was found to be completely adequate to the point it surprised the authors. RS in those Atari game experiments achieved high-scores that compete with Deep Reinforcement Learning models. Here, however, it fails completely. It is perhaps suggestive on the different nature of the domains, and how and where randomness can be useful and to what extent. Thus, in this experiment, RS serves as a lower-bound and SGD as an upper-bound (in terms of convergence speed).

TABLE 16 RESULTS OF EXPERIMENTAL SET 2: MINOR ABLATION STUDY

Name	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
LS	0.0165	100	0.6760	84
LS w/o Score Decay	0.5972	78	0.6548	76
GS	3.0430	14	3.0656	13

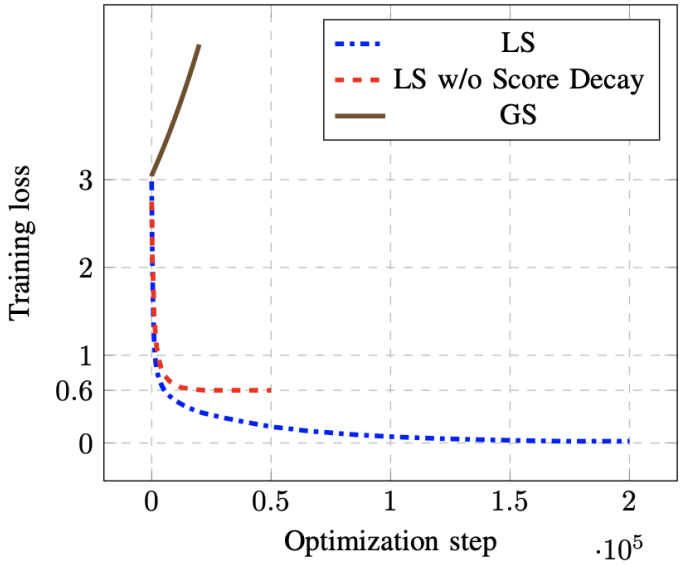


FIGURE 18 LOCAL SEARCH IS BETTER WHEN COUPLED WITH SCORE DECAY. GLOBAL SEARCH DIVERGES.

The second experimental set shows the important effect of dimensionality and Score Decay. It is clear from Figure 18 that performing a Global Search (i.e. a search in all possible directions simultaneously) is not bountiful. Thus, that algorithm terminated quickly after 10,000 steps. In addition, it is also clear that decaying the score has a large effect on convergence. In the early stages, using score decay has no significant bearing on convergence speed. However, as optimization advances, it is critical. The optimizer stagnates without it, and is terminated after 50,000 steps. Recall that score decay was introduced to help the optimizer overcome local minima and allow it to explore the search

space more actively. Its effectiveness may suggest that the search space has deep local minima.

TABLE 17 RESULTS OF EXPERIMENTAL SET 3: VARYING SEARCH DIMENSIONALITY *S*

S	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
200	0.3696	88	1.0136	78
50	0.584	99	0.7312	82
25	0.0165	100	0.6760	84
10	0.0223	100	0.6360	83

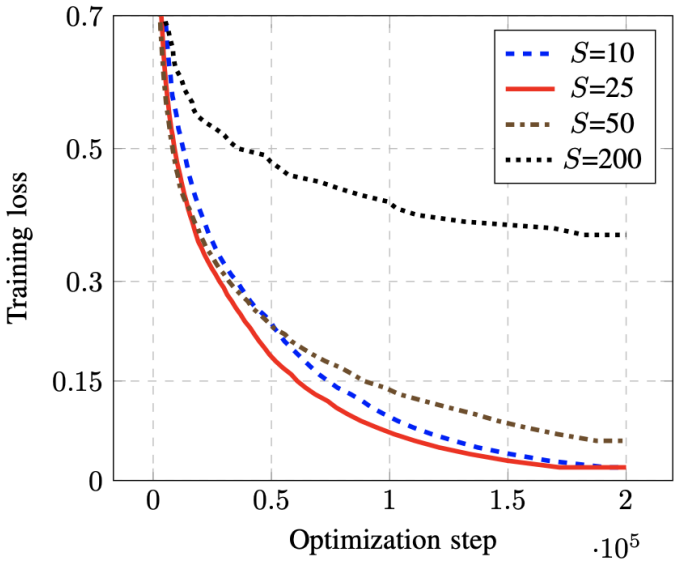


FIGURE 19 RESULTS SUGGEST THERE IS AN OPTIMAL RANGE FOR THE NUMBER OF SEARCH DIMENSIONS S, TOO HIGH OR TOO LOW DEGRADES PERFORMANCE.

From set 3, the effect of varying the number of search dimensions is clear, as demonstrated in Figure 19. First a note about the figure. That figure was cropped on the Y-axis so as to zoom-in on the bottom part. In the cropped-out part, the optimizer quickly descends as in Figure 17 and Figure 18. There is not much difference between the competitors, i.e. little to no information. Thus, the figure is cropped and zoomed-in so that the minute differences can be easily distinguishable.

Increasing the number of search dimensions (as seen when $S = 200$) moves the LS algorithm to behave more like Global Search, and thus diminishes its value. This effect further confirms the hypothesis about search dimensionality and its bearing on the search process, and the effectiveness of LS in this case. Diminishing the number of search dimensions (as seen when $S = 10$) also is not helpful. As the number of search dimensions gets smaller, the "impact" each guess has on the overall optimization is reduced. Consider the pathological case of a single search dimension. The optimizer would require a tremendous number of optimization steps to converge. This suggests that there is an upper and lower bound on the number of search dimensions to achieve optimal results with LS.

TABLE 18 RESULT OF EXPERIMENTAL SET 4: USING ACCURACY INSTEAD O LOSS AS LEARNING SIGNAL.

Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
2.7539	74	2.9072	68

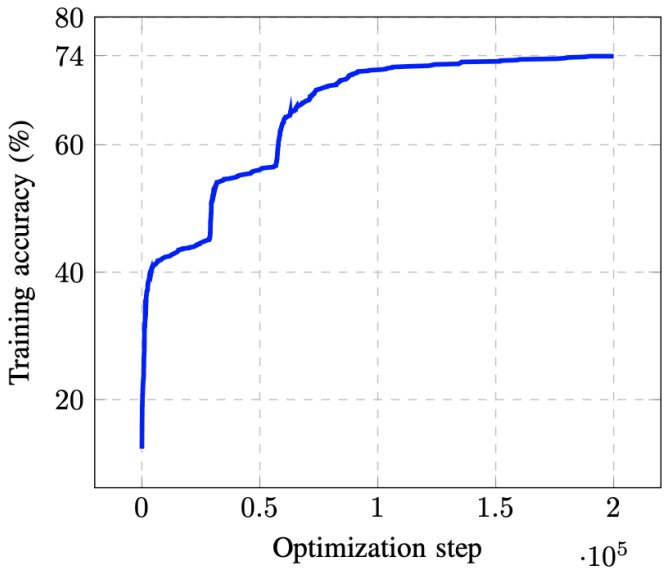


FIGURE 20 WHAT IF YOU DON'T HAVE A LOSS FUNCTION? LS CAN TRAIN THE CNN USING ONLY ACCURACY AS A LEARNING SIGNAL.

Finally, in the fourth experimental set the idea of training a neural network without a loss function is tested as seen in Figure 20. Instead of using categorical cross-entropy as was with all other experiments, the training accuracy itself was used as a learning signal. It can be seen that the network does in fact learn, it achieves 74% accuracy on the training set. And as shown in Table 18 the learnings are validated as it achieves 68% on the validation set.

The accuracy metrics are not as good as those achieved when a loss function is used for training. However, this experiment does indicate the plausibility of training without any loss function. Furthermore, it demonstrated that the hypersurface created by the loss function, i.e. loss hypersurface, is easier to navigate than the accuracy hypersurface. Remarkably, from Table 18, the both training and validation losses are quite high compared to the accuracy figures. This suggests that loss space is not the only representative space of the task. In other words, the task can be solved in other spaces which do not necessarily conform in topology to loss space. Also, there noticeable "jumps" in the graph. This suggests that learning through accuracy is sensitive to values of a small subset of $\boldsymbol{\theta}$.

Conclusions

The Local Search algorithm was used to train a convolutional neural network with 5 Million parameters on the FashionMNIST dataset. Due to resource constraints and the number of experiments performed only 10% of the training data was used (5,000 images). The optimizer managed to achieve 100% training accuracy, and 84% validation accuracy on the entire validation set (10,000 images). LS outperformed SGD in terms of final

accuracy and loss metrics. However, SGD was found to be up to 50X faster than LS. In addition, Random Search was found to be completely ineffective in training such a network, a departure from recent results in the deep reinforcement learning community suggesting otherwise.

This study, in addition to those findings, presented a minor ablation study and a survey on a significant hyper-parameter S (the number of search dimensions). The ablation study demonstrated the need for the features used in LS. Also, the intuition behind using LS has been confirmed when contrasted by the case of Global Search and generally high number of search dimensions S. It is certainly better to take random guesses in only a tiny subset of the entire search space. When S is too small performance worsened, which suggests there is an optimal range for the number of search dimensions S.

Finally, the case for training neural networks without a loss function has been presented. The CNN was trained to 74% training accuracy and 68% validation accuracy. Remarkably, those accuracy metrics coincided with relatively high loss metrics (2.75 and 2.9). This suggests that the optimization hypersurface created by accuracy is topologically different from that of categorical-cross entropy loss. Thus, training plausibility using accuracy was demonstrated, with an additional insight as a byproduct.

The experiments and approach presented in this study may hopefully act as a steppingstone towards further improvements in derivative-free single-candidate optimization algorithms for neural network training. The task of multi-class learning, the $5x10^6$ parameters and the use of FashionMNIST dataset all make the experiments here more applicable and

representative of modern deep learning tasks. Though LS is unlikely to be a competitor to SGD at this stage, it may be used in tandem with it. In cases without loss functions or without differentiability, LS can be used as a go-to alternative to the researcher.

This experiment constitutes an introduction of Local Search to successfully train a convolutional neural network with more than 5-Million parameters. The LS algorithm still requires further studies in order to fully understand the manner in which it works under different training dynamics, and how it can be effectively used.

In particular, it is desirable to understand how the tuning of the hyperparameters of the task and the trained network will affect training performance and convergence. For example, in what way does the network architecture affect training performance when using LS? In addition, how will LS perform solved Deep Reinforcement Learning tasks?

Once these aspects are studied and understood, LS algorithm and its derivatives can start to act a replacement for the traditional SGD-based training approach. As such, the researcher can have multiple tools to train her network with and be able to select the best tool for any scenario.

Experimental Study 5: Examining Hyperparameters When Training Neural Networks Using Local Search

Introduction

The principles relied upon in training neural networks have remained largely the same for decades. Specifically, the use of derivative-based optimization algorithms is by far the most common technique. Stochastic Gradient Descents (SGD) and its variants, e.g. ADAM [17], coupled with the Backpropagation (BP) algorithm [78] are staple ingredients for DNN training. As with everything, SGD and BP have limitations and challenges associated with their use. For instance, relying on derivative- based techniques requires the network architecture to be end- to-end differentiable. Also, there are tasks that cannot be solved directly such as Hard Attention [81]. Furthermore, the training process itself is an optimization of a proxy function usually called the "loss" function. If the loss function is not representative of the task, then the optimization process will fail. Thus, for every task, a representative loss function needs to be found, usually by an expert. Once the loss function is found, it can then be used by members of the respective community. An example of this is Categorical Cross-entropy which is commonly used in Image Classification tasks.

The limitations of SGD+BP motivate the search of alternative options to train DNNs. Such options include derivative- free optimization algorithms. Derivative-free methods need not compete directly with SGD or replace it. They simply can act as yet-another-tool or

alternative for researchers to use. Since they don't require a loss function, derivative-free methods can make DNN training more accessible to non-experts on novel tasks.

Regardless of the optimization algorithm, the number of hyperparameters involved in training DNNs is typically quite large. An exhaustive search is thus prohibitively costly. The Deep Learning community in general has used standard practices to select those hyperparameters. In a typical scenario, experts would try many hyperparameter configurations, e.g. manually or through a grid-search, and publish their best-performing ones. Those published configurations then become standard practices until they are replaced by even better- performing ones (state-of-the-art). Examples of hyperparameters include the number of layers, the configuration of each layer and the choice of non-linearity (activation function).

Here is a case where the choice of hyperparameters (number of layers, type of layers, etc...) was key. A likely source of improvement in benchmark tasks, e.g. ImageNet, is innovative neural network architecture. Note the improvement introduced by moving from a VGG-16 architecture [68] to a ResNet architecture [100] in Image Classification tasks. A major argument for ResNet was the introduction of residual connections as means of accommodating the nature of training through error backpropagation. By paying close attention to the training dynamics of SGD+BP, ResNet architecture improved upon the state-of-the-art in Image Classification.

This experimental study features a small-scale study on the choice of 3 network hyperparameters in regards to training with Local Search (LS) [122]. Local Search is a

rather dated optimization concept. However, it has been recently used to train neural networks successfully for image classification, albeit it was significantly slower to converge than SGD. For comparison, SGD will be included in the study to represent derivative-based optimization algorithms. LS is a single-candidate optimization algorithm, just like SGD, which makes the comparison fairer and more relevant. The study should reveal the effect, if any, of changing the selected hyperparameters.

Related Work

There seems to be a stagnation in innovation in the fundamentals of neural network training. This has also been remarked upon in [117] and [118]. There are creative variations in the training process, such as Generative-Adversarial Networks [96]. However, the fundamentals remain constant. There is a reliance on the combination of SGD variants, e.g. ADAM, and Backpropagation. Reliance on SGD+BP means that only derivative-based, single-candidate optimization techniques are investigated. Innovations such as Dropout layers [123] are thus also primarily geared towards networks trained under those principles.

Despite this, the interest in alternative methods to train neural networks seems to be rising. For example, evolutionary algorithms have been used in [75] and [74] to train networks in the field of Reinforcement Learning. The authors in those papers used Genetic Algorithms and Evolution Strategies to train network on to play Atari games. While being derivative-free, those methods are multi-candidate based. This means that at each optimization step, the algorithm performs many evaluations of the loss function. Both papers used a relatively high number of populations (on the order of $10^2 - 10^3$). Thus, the number of function

evaluations performed is quite large if directly compared to single-candidate based methods like SGD.

There are efforts to replace BP as well. Authors in [118] and [117] presented alternatives to BP called Decoupled Neural Interfaces (DNI) and Kick-back. Both methods were tested using datasets such as MNIST and CIFAR-10. The methods performed competitively compared to SGD+BP. However, both are also still in their infancy and thus warrant further work as noted by the authors, particularly Kickback.

Otherwise, there is no evidence to suggest significant interest in single-candidate, derivative-free methods to train neural networks. There were attempts, however, in the 1990s as in [119] and [120]. Those attempts, though promising at the time, are not representative of modern deep learning tasks. The networks used are too small and don't contain millions of trainable parameters as is common today. Thus, we find ourselves in the uncommon position of investigating single-candidate derivative-free optimization methods to train deep neural networks.

Otherwise, hyperparameter optimization is a lively field of study. We see examples of many recent works investigating new methods to optimize hyperparameters, particularly for neural networks. Some works use Reinforcement Learning for hyperparameter optimization e.g. [124]. A more popular technique is Bayesian Optimization as in [125] and [126], however. The study in [127] conducted 20,000 experiments using the authors' Learning Curves method. It was found that their proposal both improved and worsened performance, depending on the scenario at hand. It is a sizable undertaking by the authors

of that studying because of the relatively large number of experiments. In addition, researchers from large corporations such as Google and IBM also study the topic of hyperparameter optimization, e.g. in [128] and [129]. Furthermore, even large government labs such as the renowned Oak Ridge National Lab (ORNL) study this problem [130].

Without a doubt, the topic we study of hyperparameter optimization is important and relevant for all who take interest in deep learning and neural networks. Given the scale of this study, there is no need to use any method proposed above. These methods, e.g. Bayesian Optimization, are most suitable when the sample complexity is high and when each sample is costly to acquire. For those reasons, this study will rely on performing the hyperparameter examination manually by hand.

Experimental Setup

The LS and SGD algorithms are tested on the FashionMNIST dataset. It is a modern dataset that is representative of modern Deep Learning tasks, especially in Computer Vision. It has the same construction as the original MNIST dataset, however the classes are much more challenging. Given the computational constraints and the number of experiments performed, only 5,000 randomly-sampled images are used from the training set. However, all the test set was used (10,000 images). Another reason to use a subset of the training set is the relatively long optimization horizons of the experiments. The LS algorithm can take 10^5 iterations to converge. If the entire set is used, the wall-clock time it will take to perform all the experiments would be prohibitively long.

Mini-batch training is only possible when SGD is used, and not LS. SGD can estimate the gradient from the mini-batch, while LS requires a static loss surface. Therefore, for fairness we used batch-training only, i.e. put all the training data as one batch. This way LS and SGD can be directly compared with no variation in training processes.

The tests are conducted on a conventional VGG-like CNN with the following architecture: 2 convolutional layers with16 and 32 filters respectively, followed by maxpooling, then 2 convolutional layers with 64 and 128 filters respectively, followed by maxpooling and finally 2 fully-connected layers of size 768 and 10, respectively. The default activation function used is the hyperbolic tangent function. The loss function used is Categorical Cross-entropy. Across all experiments this same general architecture is used, with only the hyperparameter in question being varied. The tests were conducted using the PyTorch platform [104].

Experimental Sets

The experiments are divided into 3 experimental sets, where each set tests a particular hyperparameter. The first set examines the effect of varying the number of parameters of the neural network. It is interesting to assess whether, and how, this variable affects the training process. The sizes of the CNN layers are varied to achieve the number of parameters desired. In this set 10 experiments are performed, split between SGD and LS.

The second set tests the performance of the algorithms against challenging weight initialization schemes. It is well-known that neural network training is affected by the weight initialization scheme [103]. In this test, the aim is to gauge the effect of weight

initialization while training with LS. Again, for comparison, SGD is included. In total, there are 8 experiments split between both algorithms. The Uniform scheme has limits in the range [-0.1, 0.1], the Normal scheme has standard deviation 0.1, the Sparse scheme has 0.9 sparsity ratio.

The final set tests performance under different activation functions. Similar to the weight initialization, neural network training is known to be affected by the choice of activation function [103]. It is important to assess how the choice of activation function affects training when LS is used. This should allow proper selection of activation function, and perhaps reveal insights about LS dynamics. In total, 3 functions are used for both SGD and LS yielding 6 experiments.

Overall, a total of 24 experiments are performed across the 3 experimental sets. In all the tables, the configuration with the best test loss is typed in boldface.

Results and Discussion

Network Parameterization

The first set examines the effect of varying network parameterization. It is shown from

TABLE 19 AND

Table 20, as well as Figure 21 that the performance of the LS algorithm degrades with the reduced number of parameters. This degradation, while strong, doesn't render the algorithm ineffective or useless. It can be seen that the network still learns to a significant degree as proved by the test loss.

TABLE 19 VARYING NUMBER OF TRAINABLE PARAMETERS (LS)

Parameters (10^3)	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
5200	0.0165	100	0.6760	84
2500	**0.0999**	**98**	**0.6104**	**82**
440	0.2064	93	0.6556	80
122	0.4231	85	0.7176	79
41	0.6299	79	0.8529	74

TABLE 20 VARYING NUMBER OF TRAINABLE PARAMETERS (SGD)

Parameters (10^3)	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
5200	**0.09**	**97**	**0.4024**	**86**
2500	0.1710	95	0.4176	86
440	0.2198	94	0.4068	86
122	0.2673	92	0.4288	84
41	0.3003	90	0.4336	85

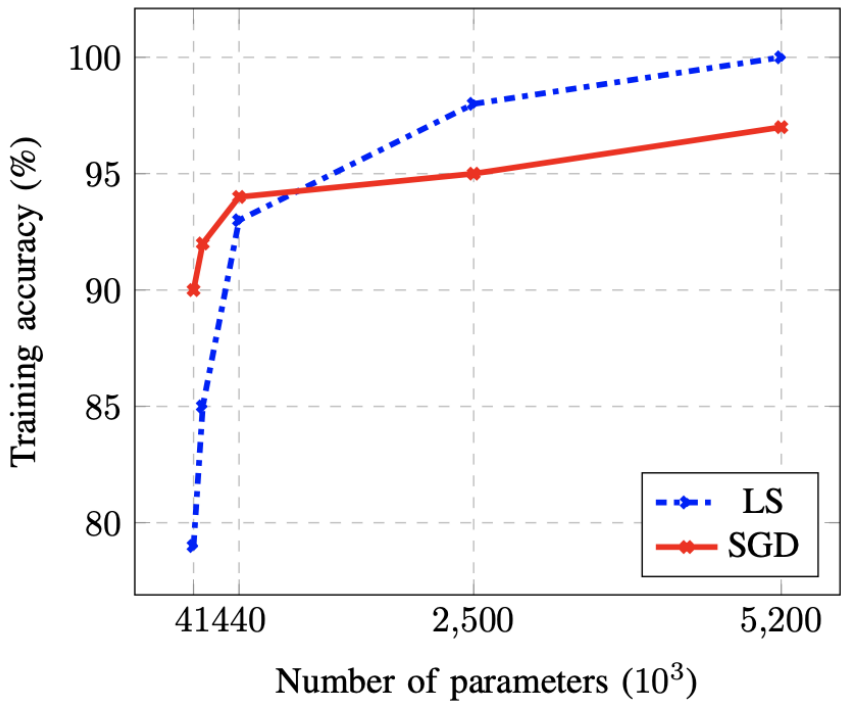


FIGURE 21 LOCAL SEARCH IS SENSITIVE TO NETWORK PARAMETRIZATION, WHILE SGD IS RESILIENT. DATA IS ACQUIRED AT 41, 440, 2500 AND 5200 ON THE X-AXIS.

In the best case, where the network has $2.5x106$ parameters, a test of 0.6 was achieved. In comparison, at the worst case when the network had $41x10^3$ parameters, the test loss was 0.85. The difference is only 0.25 between the worst and best cases. This translated into a reduction in test accuracy from 82% to 74%. Again, while appreciable, a 12% reduction in test accuracy doesn't negate the fact that the network has learned.

There could be an argument that the LS algorithm has not converged. If, however, Figure 22 is consulted, then it is evident that the algorithm has indeed converged to 0.63 for training loss. This means that regardless of the number of optimization iterations, the algorithm is unlikely to improve. What the results seem to suggest is that training is appreciably degraded as the network parameterization is diminished.

In contrast, SGD doesn't suffer from a similar degradation when the network parameters are reduced. SGD was able to train the network to 86% test accuracy in the best case, with 5.2x106 parameters and to 84% test accuracy in the worst case. Thus, there is no appreciable difference in test accuracy. In test loss as well the difference between the best and worst case is 0.03. This result demonstrates that SGD is not sensitive to network parameterization. It works equally well in both high and low-dimensional search spaces.

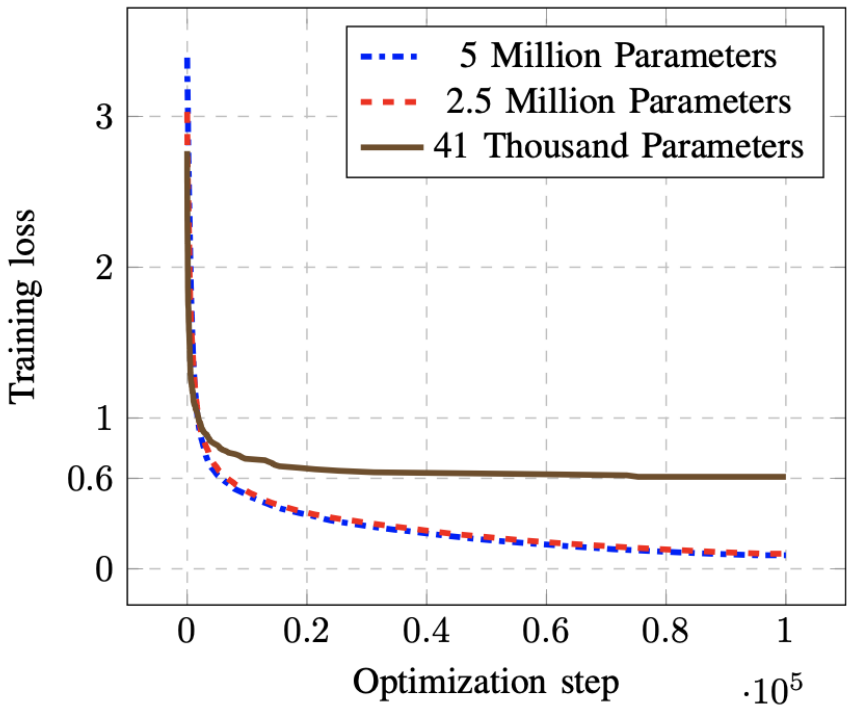


FIGURE 22 TRAINING PERFORMANCE DEGRADES WHEN NETWORK PARAMETERIZATION IS LOW, WHILE DOUBLING THE SIZE OF A LARGE NETWORK HAS NEGLIGIBLE EFFECT.

Both algorithms benefited, albeit negligibly, from the increased number of parameters. SGD trained both the 5.2 Million-parameter network and the 41 Thousand-parameter network to almost the same test accuracy. The result shows that the FMNIST image classification task is solvable with the number of parameters being on the order of 10^4. Recall that the network used in experimental sets 2 and 3, i.e. the default network, has 2.5 Million parameters. This default network is thus over-parameterized.

This is somewhat remarkable since we would expect that a lower number of parameters corresponds to lower number of search dimensions, i.e. smaller search space. The smaller search space ought to have made the optimization task, i.e. finding the best set of weights,

easier for Local Search. Evidently, the contrary is true. The LS algorithm leverages the over-parameterization of the network.

We attempt to provide an explanation by considering how the LS algorithm conducts optimization. LS randomly samples a batch of size S (where S is 25 in all the tests in this study) from the parameter vector $\boldsymbol{\theta}$, and adds noise $\boldsymbol{d}$, where $\boldsymbol{d} \sim \mathcal{U}[-0.1, 0.1]$ is a uniformly-sampled noise vector. If the applied noise yielded an improvement, then the solution is kept. Otherwise, the candidate solution is discarded. It seems that increasing the network parameterization increases the number of equally-good solutions in the search space. Consider when two solutions, i.e. neural networks, A and B represented by $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$ (where $\boldsymbol{\theta}$ is the parameter vector representing the neural network's weight configuration) achieve the same test accuracy but are not identical, i.e. $\boldsymbol{\theta}_A \neq \boldsymbol{\theta}_B$. This means that both solutions A and B are equally-good. Since they are not identical, they occupy different locations in the search space (i.e. weight space). Thus, the search space features two equally-good points in terms of evaluation metric (test accuracy in this example). Expanding on this example, we hypothesize that over-parameterization increases the number of "equally-good" locations in the search space. In other words, over-parameterization alters the geometry of the search space favorably to the optimization process.

Following this line of thought, there is a greater chance for the algorithm to "luck out". The LS algorithm has a higher chance of picking important parameters and applying useful noise (since the number of equally-good solutions is greater). It can be seen from Figure

22 that this benefit saturates. Doubling the network size from 2.5 Million parameters to 5.2 Million parameters didn't significantly affect training performance.

Weight Initialization Scheme

The second experimental set tests the effect of varying the weight initialization scheme. It is shown from Table 21 and Table 22 that both algorithms are fairly resilient to the choice of initialization scheme. However, it should be mentioned that we did not apply adversarial initializations. For example, we didn't test how the algorithms would perform in case of unbounded random initialization. The rationale behind this choice was to test the effect of practical, realistic scenarios not adversarial ones. LS trains the networks to 0.48 and 0.61 in the worst and best cases, respectively. This 0.13 difference in test loss corresponds to only 2% decrease in test accuracy.

TABLE 21 USING DIFFERENT WEIGHT INITIALIZATION SCHEMES (LS)

Scheme	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
He Normal	0.0999	98	0.6104	82
Normal	0.1652	95	0.5724	83
Sparse	**0.2141**	**93**	**0.4762**	**84**
Uniform	0.213	93	0.4936	84

TABLE 22 USING DIFFERENT WEIGHT INITIALIZATION SCHEMES (SGD)

Scheme	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
He Normal	**0.1710**	**95**	**0.4176**	**86**
Normal	0.087	98	0.5712	83
Sparse	0.3823	87	0.47	83
Uniform	0.2854	90	0.4548	84

Similarly, SGD trains the networks to 0.57 and 0.41 test loss in the worst and best cases, respectively. This translates to only a 3% difference in test accuracy. If one applies careful examination, it can be remarked that SGD is slightly less resilient than LS in the case of Sparse initialization.

Interestingly, the He Normal initialization yielded the highest test loss in case of LS and also the lowest Training Loss. For SGD, He Normal yielded the lowest Test Loss. It was odd to find that the same initialization yields the worst and best Test Loss for LS and SGD, respectively.

Choice of activation Function

Finally, the third set tests the effect of varying the activation function. From Table 23 and Table 24 it is shown that again both algorithms are fairly resilient to this hyperparameter. Similar to the previous set, only practically-usable activation functions were tested for the same reason outlined earlier.

The best test loss for LS was 0.55 and the worst was 0.67. This 0.11 difference corresponded to a 2% reduction in test accuracy. Similarly, SGD trained the network to a 0.42 test loss in the best case and 0.48 in the worst. This 0.06 difference in test loss translated to a reduction of 1% in test accuracy.

TABLE 23 USING DIFFERENT ACTIVATION FUNCTIONS (LS)

Function	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
Tanh	0.0999	98	0.6104	82
ReLU	**0.1881**	**94**	**0.5556**	**83**
ELU	0.2092	92	0.6780	81

TABLE 24 USING DIFFERENT ACTIVATION FUNCTIONS (SGD)

Function	Training Loss	Training Accuracy (%)	Test Loss	Test Accuracy (%)
Tanh	**0.1710**	**95**	**0.4176**	**86**
ReLU	0.0864	98	0.4844	85
ELU	0.0614	99	0.4532	86

Again, it is notable that the best choice for LS (Tanh) was the worst for SGD in terms of Training Loss. Both this remark and the one made in the previous experimental set suggest that SGD and LS require different, and sometimes opposing, choices for hyperparameters.

CONCLUSIONS

The experimental sets in this study comprised a small-scale investigation on how the choice of 3 hyperparameters affects neural network training dynamic under two optimization paradigms, derivative-based and derivative-free. SGD was used to represent derivative-based optimization and Local Search was used to represent derivative-free optimization; both being single-candidate based. The 3 examined hyperparameters were: the number of trainable parameters in the network (network parametrization), the choice of weight initialization scheme and the choice of activation function. In total, 24 tests were performed across 3 experimental sets.

In general, the results confirm one aspect: that there needs to be careful optimization of the hyperparameters based on which algorithm is used. The first set revealed that training performance degrades significantly under low network parameterization in the case of LS. The training accuracy was reduced by 21% and test accuracy by 10%, from the best to the worst case. SGD was resilient to this, and didn't suffer from a similar degradation. Training

accuracy decreased by 7% and test accuracy by only 2%. Note that the number of network parameters was varied by 2 orders of magnitude (10^6-10^4).

We hypothesize that over-parameterization of the network alters the geometry of the search space in such a way that makes it favorable for Randomness-based algorithms like Local Search. Thus, when designing networks to be trained by LS or similar algorithms one needs to over-parametrize, i.e. make sure there is an abundance of parameters in the network.

The second and third experimental sets revealed that the choices of weight initialization scheme and activation function are not as sensitive as the number of parameters. Performance, i.e. test accuracy, does vary but not significantly (only by 1- 3%). Notably, the best choices were different for LS and SGD.

At this early stage, it is unlikely that LS will compete with SGD in terms of convergence speed. However, it has demonstrated itself to be competitive in terms of absolute performance, i.e. training and test accuracy. For that reason, it is important to further study LS and similar algorithms and improve upon them.

The promise of derivative-free optimization of neural networks can compensate for the relatively long optimization horizon. Previously-unsolvable scenarios, e.g. when no loss function is available, may become more accessible under this optimization paradigm. This study is a step in that direction. It attempts to examine the differences in how those two paradigms (derivative-based and derivative-free single- candidate) conduct training, and are affected by hyperparameters. This aims to generate new insights, and ultimately, create new tools for the researcher and practitioner to use in their tasks.

Experimental Study 6: Solving OpenAI Classic Control Problems Using Local Search And Reinforcement Learning

Introduction

In this experiment we perform a simple validation regarding the use of Local Search (LS) algorithm in Reinforcement Learning settings. This should illuminate whether or not it is possible to use LS in domains outside Computer Vision (CV) that require a quite different training behavior. From the results of [75] and our own experiments in the previous sections, we know that training a ConvNet on a CV dataset is vastly different from training an RL agent in an RL setting. The difference is highlighted by the poor performance of Random Search in CV and its remarkable success in RL.

RL tasks vary in difficulty, and consequently the sample complexity required to solve them. It is notoriously known in the RL community that sample complexity, i.e. the number of samples taken from the environment required to solve a task, is a major obstacle [75] for RL algorithms. Sample complexity can easily jot to the order of 10^6 for many benchmark tasks [121]. For that reason, it is unreasonable to start experimenting with an introductory algorithm on complicated benchmark tasks.

As in MNIST and variants for Computer Vision, there are simple yet representative benchmark tasks for RL. To this end we resort to Open AI Gym's Classic Control problems [131]. These problems are characterized by being low-dimensional in observation space, i.e. state space, and in action space. The Classic Control problems also have short horizons, which means that the episode terminates quickly if the agent performs has a faulty policy.

Also, it does not take many steps (typically on the order of 10^3) to attain maximal rewards. These characteristics constitute an ideal test bed while also posing a non-trivial challenge for the algorithm.

It should be noted that we don't attempt to beat or meet state-of-the-art on those benchmarks. The goal here is not to improve upon SOTA but rather to validate that the LS algorithm can indeed be used in an RL setting. For that reason, we don't include other algorithms in our evaluations.

Classical Control Tasks

The LS algorithm is tested on Classical Control tasks in Open AI Gym. Namely, they are Acrobot, Pendulum, Mountain Car Continuous and Cartpole. In total, we test the algorithm on 4 tasks using the same neural architecture. Let us examine those tasks in some detail.

Acrobot

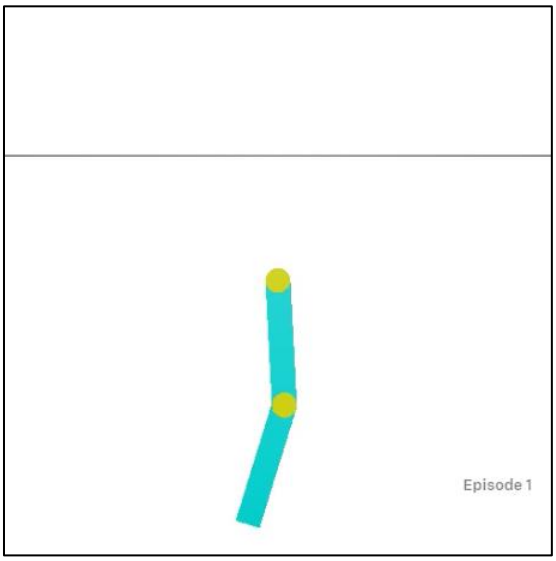


FIGURE 23 A GRAPHICAL DEPICTION OF THE ACROBOT ENVIRONMENT

The Acrobot environment features two joints and two links, where the joint between the two links is actuated. The goal is to swing the links up to a certain height. When the goal is met the episode terminates. This environment has an inverted reward scheme. It assigns a penalty of -1 with each step where the task remains unfulfilled.

There are six observation dimensions, all are bounded real numbers. The control scheme is discrete, with 3 actions corresponding to different torques (+1, 0, -1). A graphical depiction is provided in Figure 23.

PENDULUM

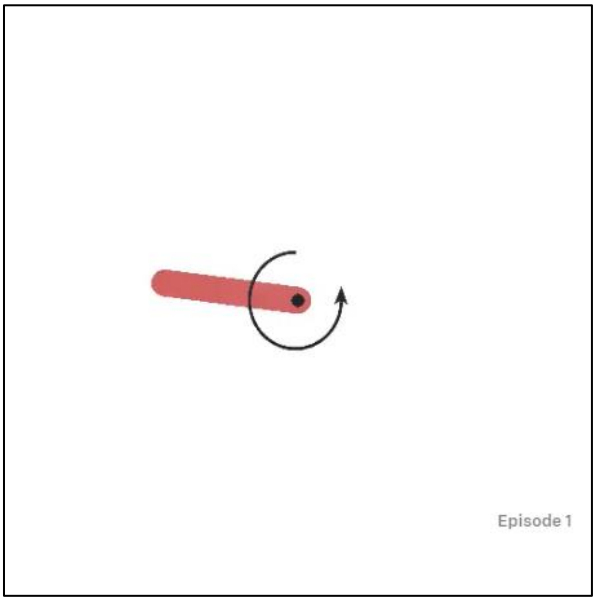


FIGURE 24 A GRAPHICAL DEPICTION OF THE PENDULUM ENVIRONMENT

The goal in Pendulum is to swing up a pendulum whose position is fixed about a pivot, and to keep it upright. The pendulum starts in a bounded random position. The environment does not simulate friction. There are 3 dimensions of observation, and one dimension of

action: the value of the effort applied to the joint (positive or negative value determines the rotation angle). Thus, Pendulum is a continuous control problem.

The reward is calculated according to a weighted distance from the reference angle (at 0^o), the angular velocity of the pendulum and the amount of effort expended. In essence, the network needs to swing the pendulum upright and keep it balance in the smallest number of steps whilst expending the least amount of effort. The pendulum starts at a bounded random location and with a bounded random velocity. A graphical depiction of the environment is provided in Figure 24.

MOUNTAIN CAR CONTINUOUS

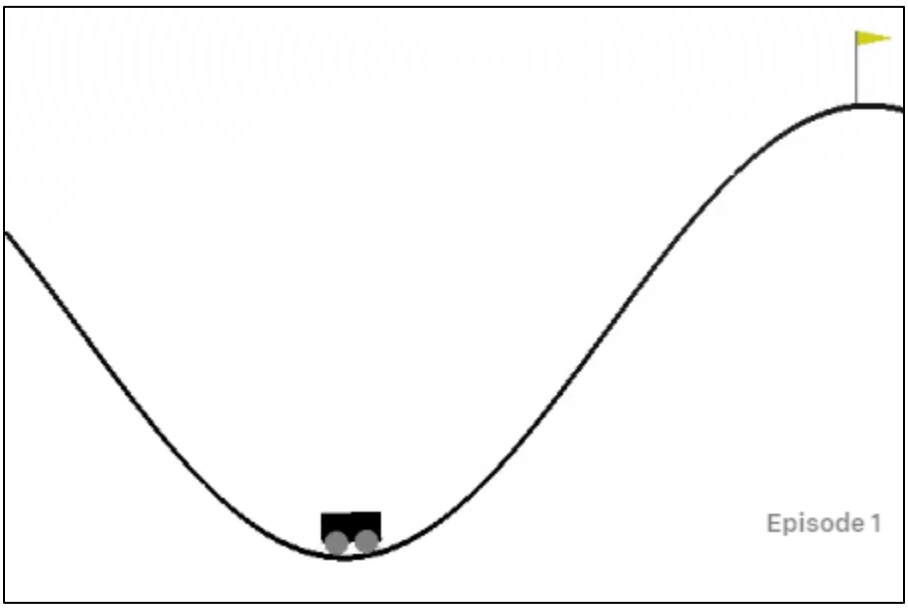


FIGURE 25 A GRAPHICAL DEPICTION OF THE MOUNTAIN CAR ENVIRONMENT

In this environment, a car is on a one-dimensional track that takes the shape of two hills. To reach the top of the right hill the car first has to build momentum by climbing the left hill. The episode terminates successfully when the car reaches the flag goal on the right

hill. The reward is calculated according to the amount of energy expended to reach that goal. The force applied in this version of the problem is a continuous real number, bounded between [-1.0, 1.0]. A graphical depiction of the environment is provided in Figure 25.

CARTPOLE

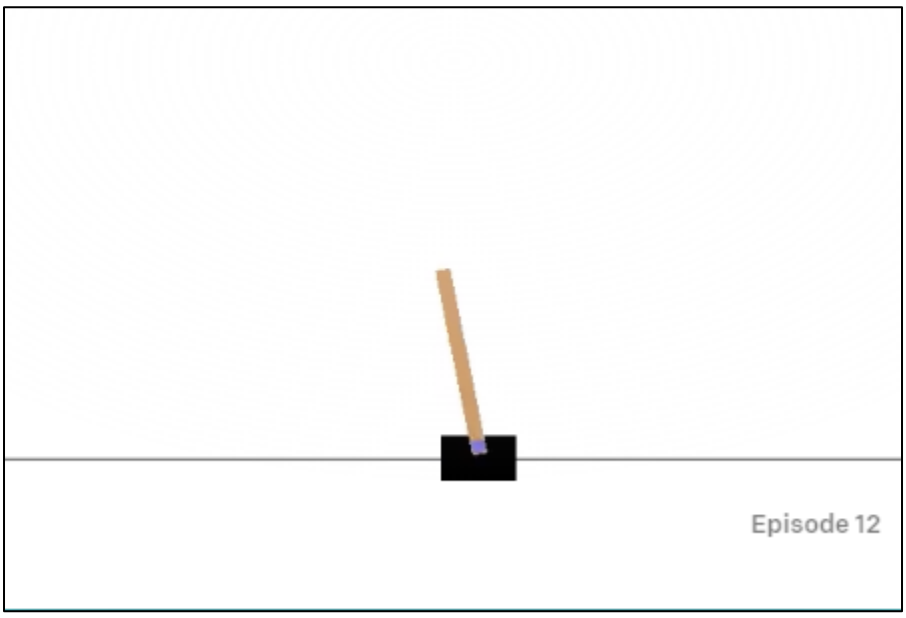


FIGURE 26 A GRAPHICAL DEPICTION OF THE CARTPOLE ENVIRONMENT

A cart is moving along a frictionless one-dimensional track. In the environment, the cart's mission is to control its movement to balance and keep upright a pole. A reward of +1 is allocated at each step when the pole is kept upright. When the pole deviates beyond a 14^{o} angle in either direction, the episode terminates. A graphical depiction of the environment is provided in Figure 26. The pole starts within a bounded random angle, and a random perturbation is applied.

SETTING

The tasks in this study were solved using an RL feedback loop. In each optimization step, the network would take control of the agent for 1 episode. Each episode lasts for N steps, or until termination either due to solving the task or reaching a termination condition (such as reaching the edge of the screen display). We set the maximum number of optimization steps to 500. Our previous knowledge about the task, and from other implementations, suggests that this task is quite solvable within this optimization horizon.

When we say that the task is "solved" in the following sections, we mean that it has successfully terminated or achieved a score that indicates that the goal of the task has been fulfilled. We do not mean solved in the context of Leaderboards where a solution is required to yield a certain average return over a certain number of episodes, typically on the order of 10^2.

RESULTS

We started with the Mountain Car Continuous problem using the Local Search algorithm on a fully-connected network of 2 layers. The number of parameters on that network was on the order of 10^6 parameters. The number of Search Directions S was 25.

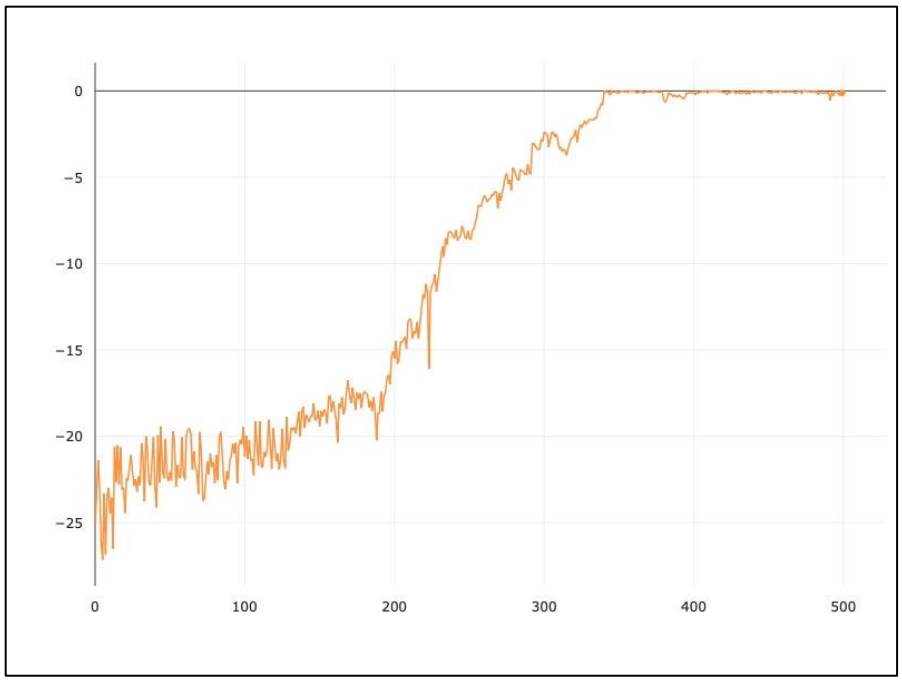


FIGURE 27 RESULT OF USING LS ON MOUNTAIN CAR CONTINUOUS TASK. THE X-AXIS IS THE OPTIMIZATION STEP, AND THE Y-AXIS IS THE FINAL REWARD. WE SEE THAT THE NETWORK LEARNS NOT TO EXPEND ENERGY (REWARD 0) BUT NOT TO SOLVE THE TASK.

From Figure 27 we see that the agent stops expending energy as the optimization progresses, but still cannot solve the task. To establish a baseline, we trained the same network using Random Search instead. Recall that RS was able to train highly-competitive agents in Atari games [75].

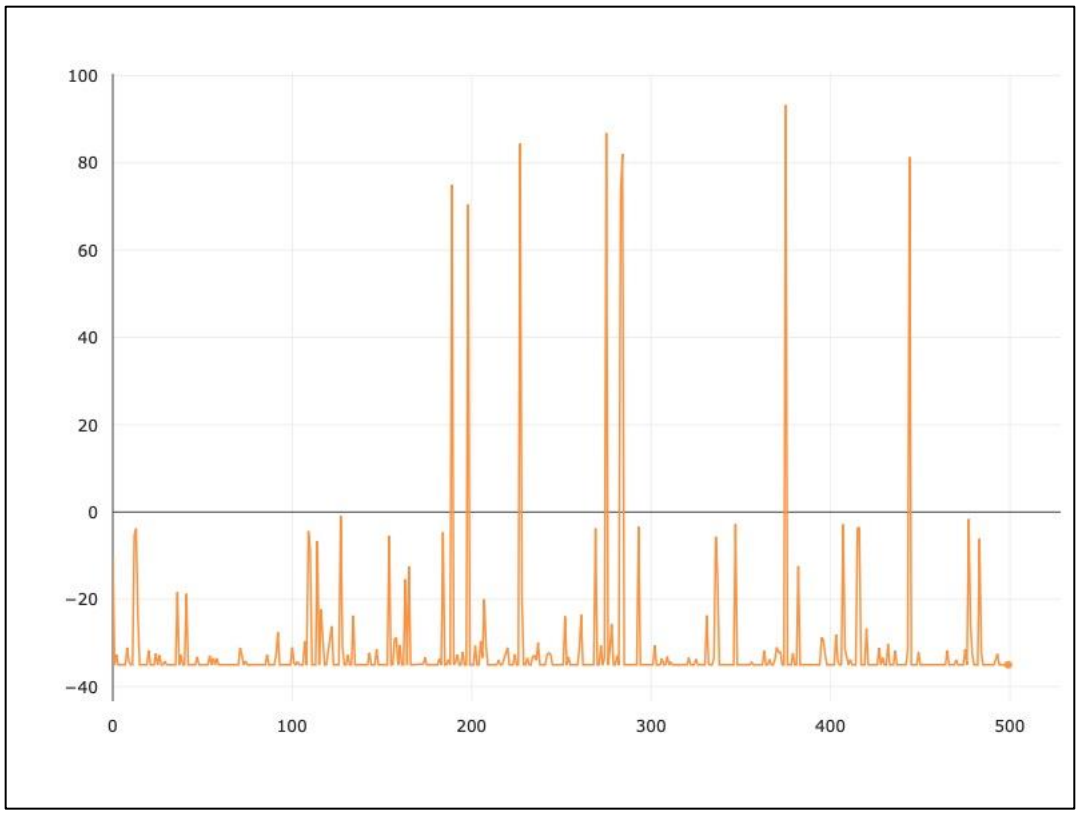


FIGURE 28 RESULT OF USING RANDOM SEARCH ON MOUNTAIN CAR CONTINUOUS TASK. RS SOLVES THE TASK.

From Figure 28 we can see that Random Search indeed solves the problem. Shown on the Y-axis are the episodic rewards tied to the optimization step. Near step 200 we see that the reward is in the positive range, which means that the task has been solved.

It was interesting to see RS solve the task and resulted in us re-evaluating what LS is doing. A key difference between LS and RS is the number of search directions S (along the parameter vector $\boldsymbol{\theta}$). The result in the figure suggested that we need to increase the number of search directions. This contrasts with what we observed when solving Computer Vision problems in the previous experiments. There we observed that decreasing the number of search directions was much more beneficial to convergence. There we also noted how RS

fails completely to train the network. This further confirms the great difference in optimizing neural networks for CV and RL.

Before we increased S, however, we wanted to assert the impact of the network parametrization on the search process (i.e. training). Recall that network parametrization had a great impact on training performance with LS in the previous experiment (using the FashionMNIST dataset).

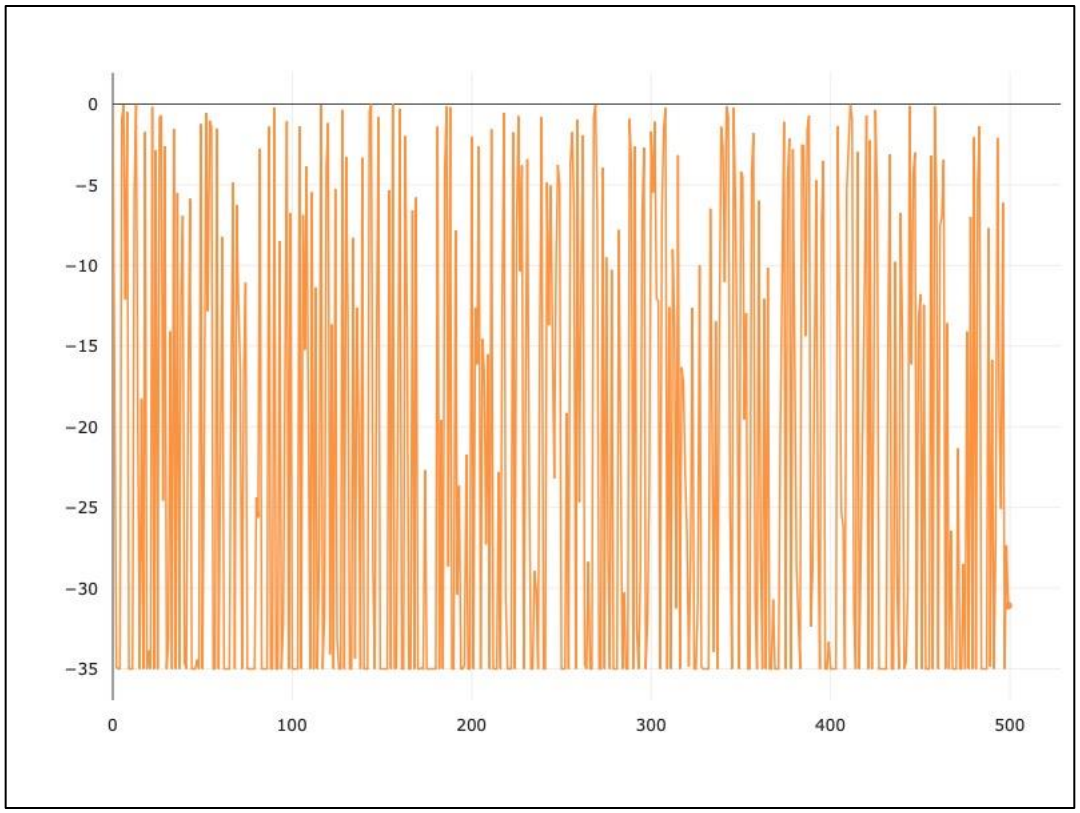


FIGURE 29 RS WAS NOT ABLE TO SOLVE THE TASK USING A SMALLER NETWORK (369 PARAMETERS).

Reducing the network parameterization from $1x10^6$ to 369 while maintaining the same network architecture but reducing the layer sizes, we re-ran the experiment using RS to

train the network. We see from Figure 29 that RS was unable to train the network. Given the simplicity of the task, we know it should still be solvable using only 369 parameters. What we conclude from this is that Randomness-based algorithms require over-parameterization of the neural networks to improve perform. We confirm that this is the case also for LS as can be seen from Figure 30.

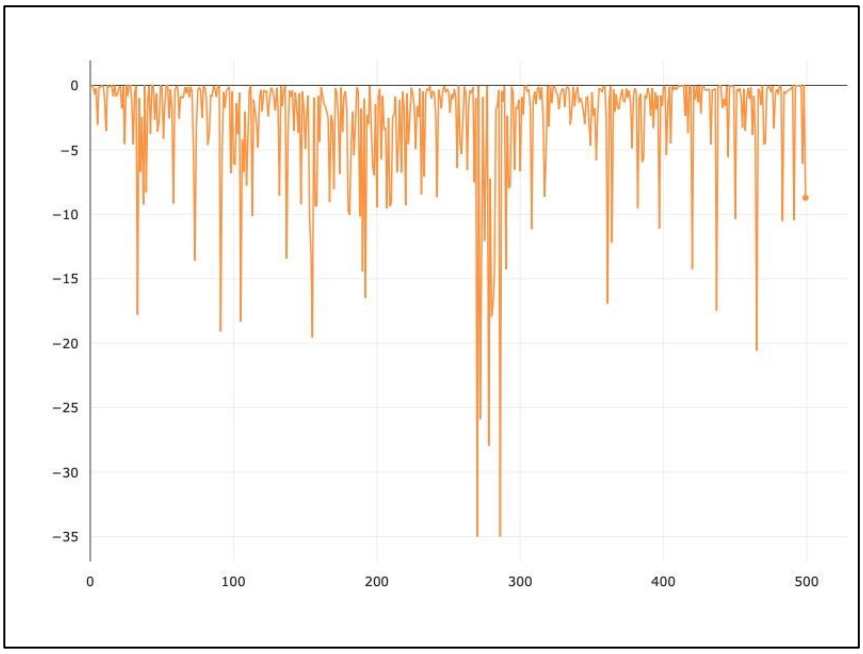


FIGURE 30 LS ALSO FAILS TO SOLVE THE MOUNTAIN CAR CONTINUOUS TASK WHEN THE NETWORK PARAMETERS ARE ONLY 369.

Using all these insights, we increased the number of search directions S to 250 first and then to 2500. We still observed that the task is unsolved. However, moving from S=25 to S=250 allowed the agent to reach the near-zero reward quicker as can be seen in Figure 31. This encouraged us to increase it further increase S to 2500. We again saw that while the task remained unsolved, the agent reached near-zero rewards quicker, check Figure 32.

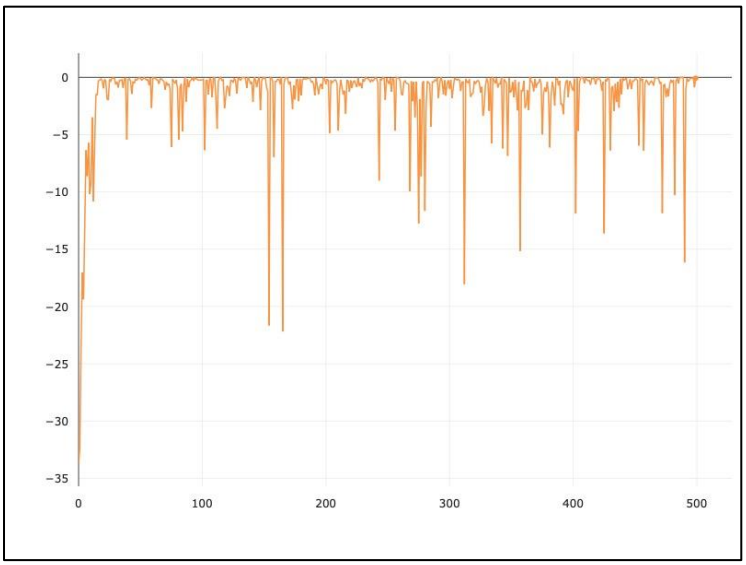


FIGURE 31 AT S=250, LS IS STILL UNABLE TO SOLVE MOUNTAIN CAR CONTINUOUS. THOUGH IT JUMPS TO NEAR-ZERO MUCH QUICKER THAN S=25.

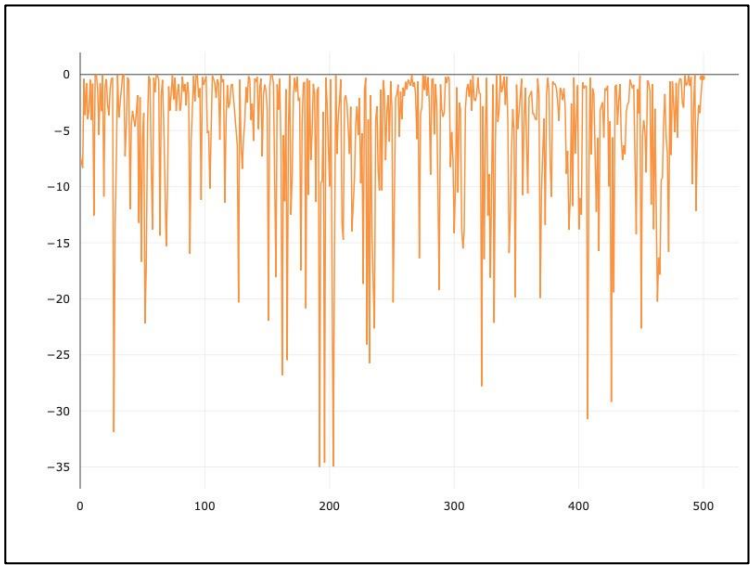


FIGURE 32 AT S = 2500, LS IS STILL UNABLE TO SOLVE MOUNTAIN CAR CONTINUOUS. IT REACHES NEAR-ZERO REWARD QUICKER THAN S=250.

These results confirmed our hypothesis that increasing S leads to better training performance for LS. We thus increased S to 250,000.

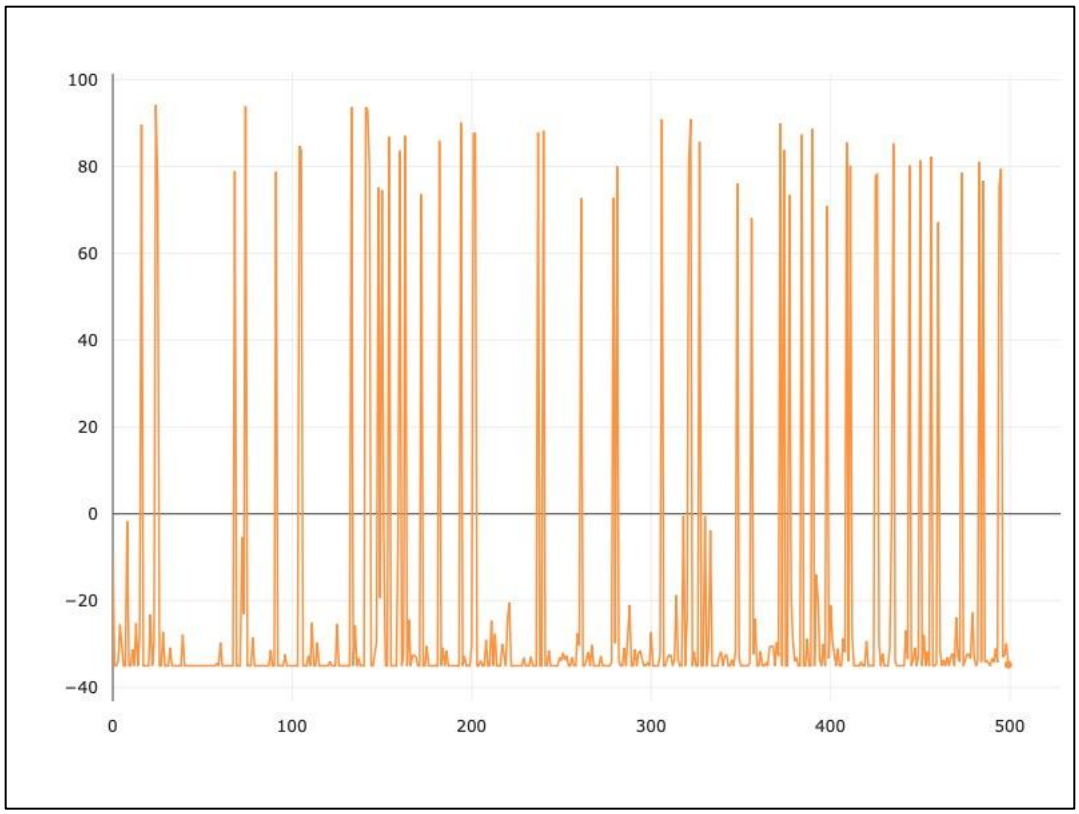


FIGURE 33 LS FINALLY SOLVES MOUNTAIN CAR CONTINUOUS TASK, S=250,000.

From Figure 33 we see that LS was finally able to solve the mountain car continuous task. It achieved high rewards, near 90. In addition, we observe that it reached this solution much quicker than Random Search did. This confirms our hypothesis about network parameterization but challenged our previous hypothesis that S needs to be much smaller than the size of $\boldsymbol{\theta}$.

Following all these insights, we now use LS under these new conditions to solve the other environments.

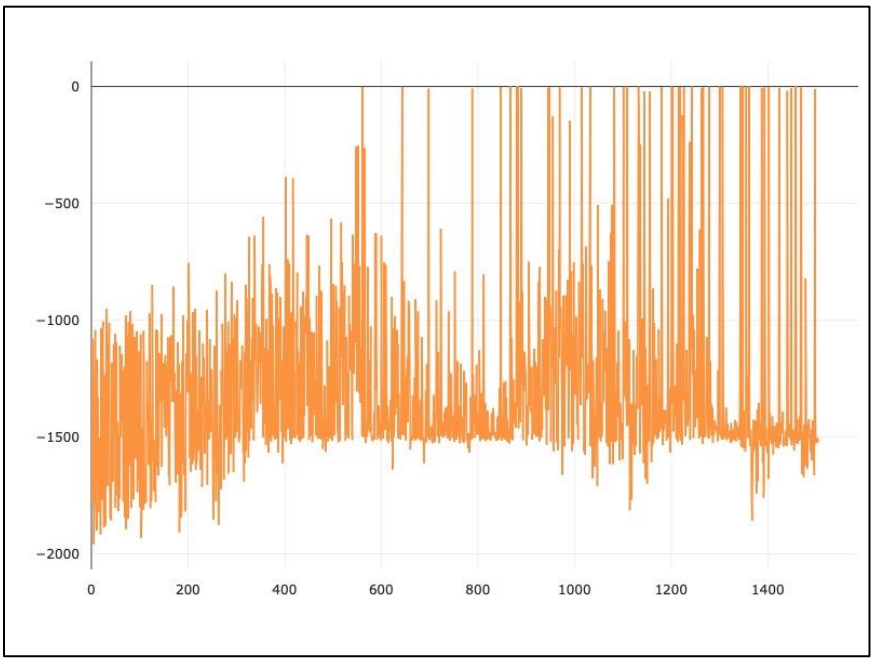


FIGURE 34 LS SOLVES PENDULUM BY GETTING REWARDS NEAR AND OVER -150.

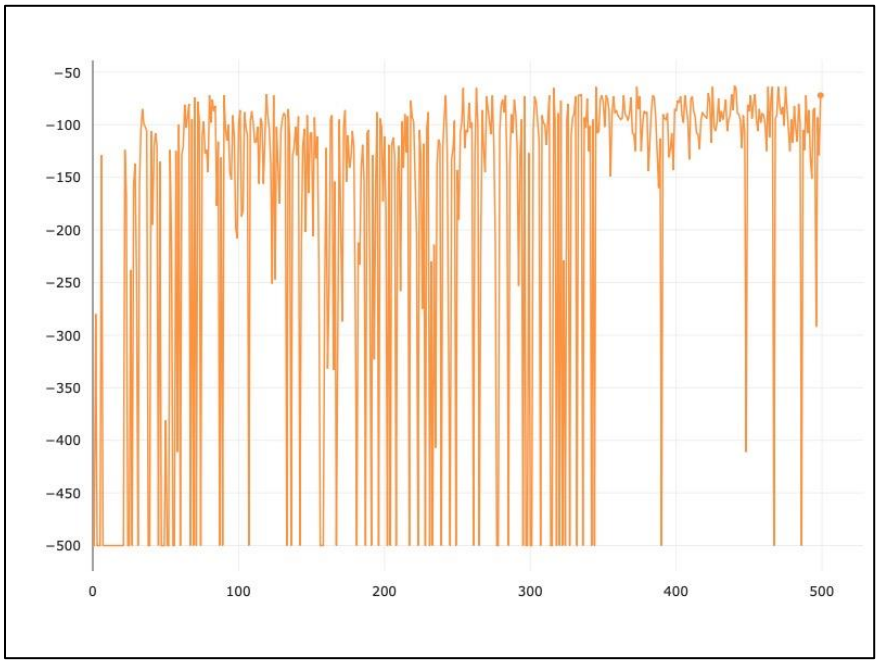


FIGURE 35 LS SOLVES ACROBOT BY GETTING REWARDS NEAR AND OVER -80.

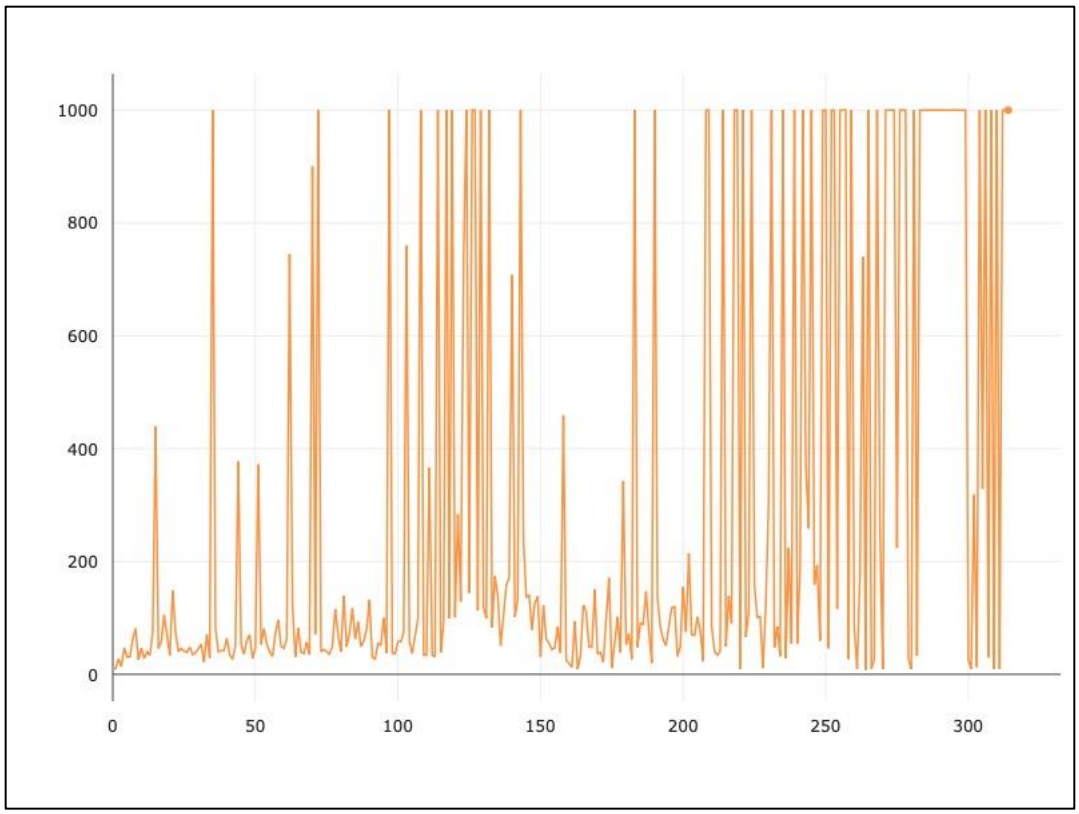


FIGURE 36 LS SOLVES CARTPOLE BY REACHING MAXIMUM REWARDS NEAR 1000.

In Figure 34 we see that LS is capable of solving Pendulum. Recall that to fulfill the task, the reward needs to be near -150 and better. We thus see that near step 600, the algorithm finds a good solution near 0. The subsequent steps are all attempts to improve that solution. We have needed to extend the optimization horizon from 500 to 1500. In Figure 35 we see that LS solves Acrobot as well. It gets rewards in the -80 range and above. Finally, we also see from Figure 36 that LS can solve Cartpole by attaining the maximum available reward of 1000.

Hence, from this section we conclude that LS can solve all Open AI Gym Classical Control problems. We confirm that the insights generated from the Mountain Car task has extended

to the other problems. It is important to note that while the tasks were solved, the performance could have been better had a hyperparameter optimization process been applied. We did not optimize the hyperparameters (such as S) because the goal of these studies is not to beat a benchmark, but to ascertain the capabilities of LS in different RL settings.

Robustness

We wanted to test whether LS can produce robust solutions. Recall that in all the Classical Control environments, there are random perturbations. Solving one instance (i.e. episode) of the problem as in the previous section may be enough as an existence proof. However, it is insufficient to produce robust neural network controllers (i.e. agents).

To produce robust controllers, the agent must be able to perform the task across almost all possible perturbations. In this regard, we expanded our RL loop so that each optimization step would not feature a single episode but many episodes (with the rewards being aggregated). The neural network will experience many perturbations throughout those episodes, and thus its true capacity and robustness will be evaluated.

While this method is effective, it is also costly in terms of resources and time. For those reasons, we choose the Cartpole task as the reference for robustness. It is a simple yet representative task for this category. Our goal in this section is to produce a robust controller for the Cartpole task. Again, the method is to increase the number of episodes per optimization step while aggregating the rewards.

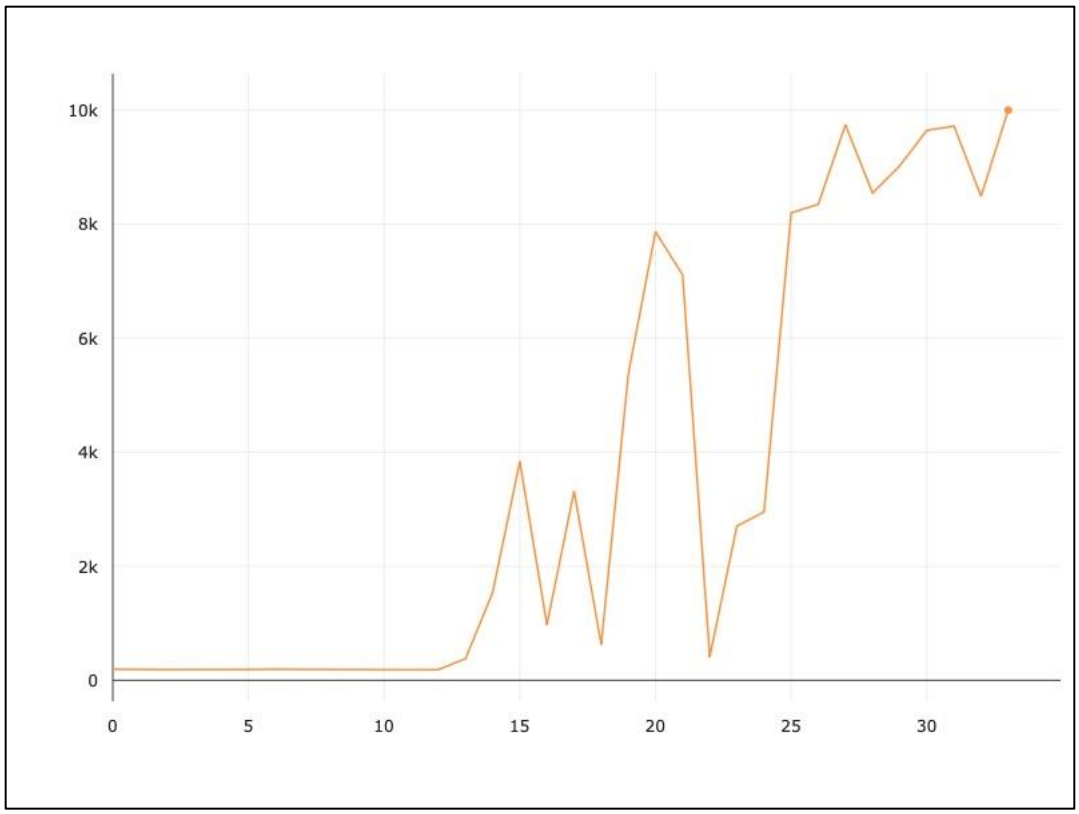


FIGURE 37 LS IS ABLE TO PRODUCE A ROBUST CONTROLLER. IT ACHIEVES REWARDS NEAR THE MAXIMUM POSSIBLE OF 10,000.

From Figure 37 we see that LS quickly reaches near 10,000 rewards, which is the maximum. The network in each step controls the agent for 20 episodes where each episode lasts for 500 action steps. Thus, the LS algorithm has successfully produced a robust controller that is able to control the agent under various perturbations.

CONCLUSIONS

Local Search was used in a Reinforcement Learning setting to solve the Classical Control tasks in Open AI Gym. Using the insights from the previous experiments, we over-parametrized the control network to be on the order of 10^6. This did indeed prove important as was confirmed afterwards. However, unlike insights from the previous experiments on

Computer Vision task (Fashion MNIST dataset), making the search direction S to be much smaller than the network parameter vector size is not a beneficial decision.

It was found that Random Search can solve the tasks. Using this insight, by increasing S, we were also able to solve the tasks using LS. This suggests S is dependent on the optimization environment or context. It may be the case that S can be adjusted via an on-line heuristic during optimization. This may alleviate the worry of optimizing S for every scenario. In addition, we were able to solve the tasks quicker than RS, which confirms LS as a better option.

Following the solution of the tasks, we also tested for the robustness of the solutions. It was found that LS can produce robust controllers. To do so, we used an aggregated-reward scheme and expanded the number of episodes per optimization step. This scheme was successful for the Cartpole problem, and we hypothesize that it would be just as effective for the other problems.

We conclude that LS is possibly an option for researchers to use in RL settings. From our results, we are encouraged to try LS on more challenging tasks such as Atari games and Mujoco 3D control environments. We provide the results in this experimental study as a steppingstone for more complex problems.

CONCLUSION

In this chapter we investigated a single-candidate algorithm called Local Search to train neural networks. The algorithm proved capable of addressing all the benchmark problems in the performed experiments. However, the algorithm also had a high sample-complexity. It is not yet ripe to test on physical robots for physical tasks. Nonetheless, the fact that there exists a single-candidate approach that is derivative-free and capable of training neural networks with millions of parameters should not be dismissed. It's a remarkable achievement to the state-of-the-art in derivative-free optimization methods. Historically and fundamentally, derivative-free methods did not perform well when the number of search dimensions was high (10^6 is considered immense). Furthermore, multi-candidate methods would generally perform much better than single-candidate due to the latter's limited capability to explore.

For those reasons, it is significant to have a single-candidate derivative-free method to train networks with millions of weights on complex, modern problems. Regarding ERL, there is a need to improve upon the current implementation of LS by first understanding its dynamics better. Future iterations of LS should be able to demonstrate their effectiveness to produce learning robots with the ERL method.

CHAPTER 6: SUMMARY & CONCLUSIONS

OVERVIEW

The topic of Robot Learning has been investigated and based upon our observations a method to train robots called Experiential Robot Learning (ERL) has been proposed. The ERL method is poised to address a gap we identified in the literature. The problem is that robots are not mature enough to be used in unconstrained environments (i.e. in the wild) because they cannot learn and thus cannot respond to new situations. Our hypothesis therefore is that the development of a methodology that permits experiential learning could allow robots to learn and therefore to succeed in novel situations.

What are the requirements on experiential robot learning that would enable robots to succeed in novel situations? ERL would need to be experiential, open ended, scalable and platform agnostic, among many other characteristics. The ERL method resembles aspirational goals which we hope robotic solutions would achieve, e.g. being Scalable. In this regard, Neural Networks (NN) provide a promising path towards achieving ERL and this dissertation evaluates this promise through a series of experimental studies.

The early experiments illuminated a problem with using Deep Learning for ERL: the need for differentiable, informative loss functions can't always be satisfied. The exploitative behavior of gradient-following is incompatible with the exploratory nature of ERL.

To address these shortcomings, we developed Local Search algorithm, a gradient-free single-candidate optimization algorithm to train neural networks. It provided good results

on the experiments performed in absence of a differentiable loss function. LS creates a viable path towards the implementation of the ERL method.

Discussion

Training NNs with Deep Learning

In the first experiment we demonstrated how to apply ERL to current Deep Learning techniques and build a robot that learns experientially. The robot was autonomously able to build and expand on a visual dictionary of identifiable objects by interacting with humans and the physical environment. The implementation obtained a high number on our Attractiveness Score (AS) scale, which rendered it a success.

The early experiments, however, demonstrated a problem with using NN (i.e. Deep Learning) for ERL: the need for differentiable loss functions & neural architectures can't always be satisfied. Moreover, the exploitative-nature of gradient-descent is not suitable to solve problems that require exploration. There are also many tasks where the optimization hypersurface is riddled with deep minima and saddle points. In cases where the gradients are inaccessible or uninformative (which is well-documented), then another family of optimization algorithms is required. The ERL method in its core is inherently exploratory, the robot is expected to "discover" solutions to novel problems on its own, experientially. After solving one problem, it is expected to solve another problem and so on. This fundamentally challenged the statistical-optimization approach of Deep Learning. Recall that the Deep Learning training technique at its conception was almost exclusively used in Supervised Learning settings on limited datasets, e.g. MNIST.

In response to these limitations, we developed 3 gradient-free algorithms called Accelerated Neuroevolution, Multiple Search Neuroevolution and Local Search. All three algorithms were developed, and validated, to train deep neural networks and provide good results on the experiments performed.

Training NNs with Evolutionary Techniques

The first algorithm, Accelerated Neuroevolution, is a primitive form of Neuroevolution with mechanisms to ensure diversity and convergence speed. Being an evolutionary algorithm, it is a member of the multi-candidate optimization family. Against a baseline Neuroevolution algorithm, AN was quicker to train a NN to solve (i.e. play well) the game of Flappy Bird. Testing AN on a real robot revealed that it satisfied ERL principles. The robot was able to discover the reward policy experientially and fulfill the simple task assigned to it. The experiment was a simple discrete control of a single joint, i.e. 1 Degree-of-Freedom (DOF). This is not unheard of in robotics, especially for introductory tasks. This experiment also featured an integration between Deep Learning and Neuroevolution, where the Perception module was trained using Deep Learning, and the Control module was trained using AN. This hybrid architecture was the first of its kind to be used in this way. Thus, the method of ERL was successfully applied to Neuroevolution methods as well. The robot again ranked high on our AS scale.

A major limitation for AN however was the fact that it is a multi-candidate optimization method. This essentially translated to a high sample complexity for robot exploration tasks. In other words, the robot would take too long to solve a problem because taking a single

optimization step requires the evaluation of an entire pool (group of candidate solutions). In addition, the benchmark of solving Flappy Bird constitutes too low of a standard to compare against.

Multiple Search Neuroevolution (MSN) expanded on the approach of Neuroevolution but employed advanced heuristics. Many mechanisms were introduced to acquire improved search behavior. While excelling at exploration, evolutionary methods are challenged by exploitation, i.e. the careful movements in Search Space to capitalize on narrow but profitable regions such as Valleys (which comes naturally to gradient-free methods). In addition, there is the curse of dimensionality. As the problem size increases, the neural network size increases (because the representation capacity required to solve the problem increases) and with it the search dimensions. This increase in search dimensions leads to a non-linear (i.e. substantial) decrease in search speed because the search space typically becomes more complex.

The advanced heuristics in MSN were found to help in this regard. The experiments performed on Global Optimization functions included a wide range of competitive evolutionary and derivative-based algorithms. MSN improved upon the convergence speed of all the competitors by at least 7X and an order of magnitude at most. Though these functions were limited to 2 dimensions, they still posed a formidable challenge as proved by the poor performance of the other algorithms. The optimization hypersurfaces of those functions are pathologically designed to be adversarial. They challenge many different properties of optimization algorithms, e.g. exploration vs. exploitation.

Furthermore, MSN trained a 5 Million parameter Convolutional Neural Network (CNN) using only 50 populations on a subset of the MNIST dataset to 90% validation accuracy. As a reference, Stochastic Gradient Descent (SGD) was included in those experiments. Training such a relatively large network with a small pool has not been achieved before.

Despite the successes, there were significant drawbacks. First, it is argued again that MNIST is not a particularly difficult problem, or representative of modern Deep Learning challenges. Second, the convergence speed of MSN was much slower than SGD. It should be noted that SGD and its variants are the backbone optimization algorithm of Deep Learning. With MSN being a multi-candidate optimization algorithm, and SGD being single-candidate, the problem of convergence compounded the sample complexity. Meaning that in order to take one optimization step, SGD needs to evaluate one sample (because it is single-candidate), while MSN needs to evaluate many. Finally, MSN featured many hyperparameters due to its reliance on advanced heuristics. This means that MSN may potentially require heavy hyperparameter tuning prior to being applied on different optimization tasks.

Training with Single-Candidate Techniques

Following the limitations of MSN, it was clear that multiple-candidate methods will not suffice for an efficient implementation of ERL. The robot cannot be expected to perform N evaluations (even if N is only 15, which is a relatively miniscule pool size) before taking a single optimization step. It will always be the case, that if we include a single-candidate method as a reference it will have a much lower sample complexity than a multi-candidate

method. Of course, the single-candidate method would need to be able to perform the task well (the task's loss function must have informative gradients).

For those reasons, the next algorithm we present is Local Search. It is a single-candidate gradient-free algorithm to train neural networks. By performing multiple evaluations from the same location, multi-candidate methods get a noisy estimate of the gradient. Despite being derivative-free, multi-candidate method can still leverage this noisy estimate information to push the search in the right direction (when appropriate). Single-candidate methods don't even have that luxury, in a general sense.

The Local Search algorithm performs well on the experiments performed. It worked in both Computer Vision and Reinforcement Learning settings. The results indicate that LS is much better than Random Search but worse than SGD in terms of convergence speed. In case of Computer Vision setting, a subset of the Fashion MNIST dataset was used. This is a much newer dataset that is representative of modern Deep Learning tasks yet simple enough to be an introductory task. LS was able to train the neural networks without any loss functions.

From these experiments, we discovered interesting insights about the search dynamics of LS under different conditions. Network parameterization as well as the number of search directions are both critical to achieving a good performance from LS. To conclude, LS provides a plausible option for implementation in robots that learn using the ERL methodology.

As a summary and for reference, we provide a comparison between the 3 techniques to train NNs in Table 25.

TABLE 25 AN OVERVIEW COMPARISON BETWEEN THE 3 NN TRAINING TECHNIQUES

Technique	Deep Learning	Neuroevolution	Local Search
Derivatives	Required	None	None
Underlying Algorithm	SGD	Varies	Random Search
Loss Function	Required	Optional	Optional
High-Precision	Required	Optional	Optional
Natural Behavior	Exploitative	Explorative	Balanced
Optimization Family	Single-candidate	Multi-candidate	Single-candidate
High Dimensionality	Handles well	Typically Struggles	Typically Struggles
Sample Complexity	Naturally Low	Naturally High	Naturally Low

SCOPE

In this dissertation we started from a very high-level view of the problem where the focus was on motivation, purpose and methodology. In this light, the ERL method was developed. We progressively delved into lower-levels. The work after the method comprised of focusing on possible applications of ERL and examining different implementations. The first experiment reflected this focus on implementation-driven research and experimentation.

However, we ended up diving into a deep, low-level view of the problem. The fact that we studied optimization algorithms of neural networks demonstrates this. More often than not, and the lack of innovation in training algorithms is evidence, research is stuck at the medium-level where it is application-driven. That we went deeper than most and generated new optimization algorithms highlights the fundamental nature of the ERL method and how it is different from everything else. The reason we went so deep is that we wanted to

adhere to ERL as much as possible. The then-current state-of-the-art (i.e. Deep Learning) was not conducive to adopting this approach.

Recently, this has started to change. In the past two years, there has been a growing interest in fundamentally changing the way neural networks are trained. Gradient-free methods seem to be gathering traction after investigations from major, world-class technology labs such as Google DeepMind, Open AI and Uber AI.

The research in this dissertation could have taken so many directions. It was decided to keep the focus as much as possible on robotics. However, it was not always feasible to do so. First, the low-level investigations warrant that the newly-developed algorithms be validated on common benchmarks before deployment on a robot. These algorithms need to be better than other algorithms in some respect, as well as overcome some of the obstacles facing the implementation of ERL. This meant that we cannot immediately test on robotic applications. Thus, the focus deviates into tasks like global optimization functions.

Second, the timeframe allocated to development of these optimization algorithms was quite limited which meant that rapid prototyping is a must. All the candidate algorithms needed to prove their worth or be discarded in as little time as possible. Needless to say, it takes a long time to iterate on an idea until the final algorithm is produced. Robotic tasks are inherently time-consuming. The robot takes time to plug-in, charge, set-up, use and store away. In addition, doing something in the real physical environment is naturally more

demanding than performing simulations. In consequence, we couldn't adhere to focusing solely on robotic applications throughout this dissertation.

If the state-of-the-art training algorithms and neural architectures were conducive to use under the principles of ERL, all this time and effort would have been saved and diverted to pure robotics research. Pure robotics research here refers to the physical robot implementations under different scenarios.

The scope pre-defined in the first chapter underlines the approach adopted throughout this dissertation. The purpose of this investigation is not to beat the state-of-the-art in any task or benchmark. If that was the case, one would simply optimize a robot to perform said task and once beaten declare that the job is done. This in fact would be the exact approach we are against since the definition of the motivation. Specifically, that hand-engineering approaches are ungeneralizable and unscalable. It would be folly to attempt to address this problem by yet-again creating and focusing on engineering a solution to a specific problem/task/application.

Hence for the entire course of study, the focus has been on general learning problems. The goal of all the robot experiments was to demonstrate the capacity of experiential learning, not to fulfill the task. We limited the test applications to performing a single task in any given experiment. If learning can be demonstrated on one task then our goal has been fulfilled, irrespective of whether the robot can solve another task(s). The tasks themselves were also designed to be straightforward and simple in nature. Yet, they were also challenging and satisfied the requirement that they couldn't be performed without learning.

This balances the time & effort spent time on solving the task, and the significance of the achievement.

In summary, we attempted to stick to practical implementations of ERL when possible and veered only when necessary. The performed experiments were chosen with deliberation to cast a balance between rapid prototyping and meaningfulness & relevance. The approach adopted in the whole of this dissertation was a top-down approach. We started from the top by making observations and developing a method. Then we ventured to apply this method using then-SOTA optimization algorithms. Finally, we delved into the low-levels and developed novel optimization algorithms which are more conducive to the afore-defined ERL method. Our method, combined with the new algorithms, provide a clear path for future research to allow robots to learn from physical experiences in the real environment.

FUTURE WORK

Thus far in the research we have reached the point of validating the LS algorithm (one of the developed gradient-free algorithms) on relatively simple tasks. The LS algorithm was developed to enable the ERL method in cases where Deep Learning approaches are not effective or possible. We demonstrated how the ERL method can be applied to a real robot application using Deep Learning approach, i.e. SGD, not LS.

The obvious direction of future work is to use LS to demonstrate ERL principles on a real robot in elaborate settings. Recall that ERL attempts to reject sim2real techniques (performing learning in simulation and then transferring the "trained" network to the real environment) in favor of learning on-the-job, i.e. in the real physical environment and not in simulation. ERL does not negate simulations entirely, but it rather attempts to stress the importance of physical learning from experience (the *ability* to learn experientially, specifically). There is a heated debate at the moment on sim2real approaches, following the impressive-yet-disappointing results from Open AI Robotics [132]. The impressive aspect is that a neural network can control a robot with unprecedented dexterity. The disappointing aspects are that the robot achieves the task (manipulating a Rubik's cube) correctly 60% of the time, and 0% at worst case, and that the robot took about 10,000 years of simulated experience to reach this level (months of wall-clock time). It leaves us wondering how many years of simulated experience it will take to learn to walk for instance. For those reasons, the ERL method may be needed so that it presents a voice that emphasizes the importance of learning from physical experience. Understandably, physical

experiments are costly in time and resources. Thus, the level of complexity of physical experiments must be scaled according to the progress made here and in subsequent work.

Following the last 3 experiments, we can surmise that the LS algorithm itself needs further study to fully understand its training dynamics. There are questions regarding the use of randomness and heuristics, and the impact those would have on convergence. There are also open questions regarding the neural architecture, and whether LS can successfully train more elaborate cells like LSTMs.

Another direction of research is to study the combination of evolutionary and single-candidate methods to form more powerful optimization algorithms. It may be the case that a single-candidate is too linear (lacking in diversity) and a pure multi-candidate approach is too costly (in terms of sample-complexity). A hybrid approach where a relatively tiny number of candidates (under 10) are used may be promising. The increased sample complexity may be compensated by a faster convergence rate.

The advanced heuristics employed in MSN algorithm may also be streamlined and used to enhance the LS algorithm. The use of heuristics may alleviate the need to tune the number of search directions, which is an important parameter as discussed in the Hyperparameter-search experimental study. This approach of employing heuristics may also improve the exploration/exploitation characteristics of LS. Ideally, this would lead to faster convergence, possibly translating into faster learning in the physical environment.

Lessons Learned

The lessons learned would require a chapter of their own should we include everything. The PhD journey is gruesome and grilling to say the least. One often fails rather than succeed. Failure becomes the norm, not success. This begs the question of how one would succeed after constant failures. In this section we attempt to capture those answers.

First and foremost, it has been noticed that the state of the art in AI and Machine Learning is constantly shifting. The bar upon which upcoming algorithms and implementations are compared to is getting higher. In those circumstances, it is rather difficult for one to innovate.

To make an innovation one must take risks, and often those risks do not pay off. In a highly-challenging environment, however, we don't have the luxury of not innovating. The shift in this thesis from gradient-based to gradient-free algorithms is one such risk. Had we not taken that risk; we would have encountered many obstacles to adhering to the defined principles of ERL.

Thus, the first lesson is not to follow state-of-the-art but rather to attempt to create your own. The second lesson is, sometimes it is simply unfeasible to implement what one proposes. The current technological offering may not be conductive to what is being proposed.

For example, in our proposal of ERL we wanted robots to learn online. However, we found that due to the current technological offering in embedded systems it is nigh impossible to

embed large deep models in training loops. It is possible to use deep models in inference loops, meaning that the agent is already trained. But training on embedded systems is not feasible today from our experience. As such, we resorted to shifting the training loops onto a desktop workstation and communicating with the robot via Wi-Fi. The workstation would capture the robot's sensory inputs, perform training, and then proceed to feed the robot the action policy.

Third lesson is to create what isn't there should there be a need for it. In all our endeavors it was quite challenging to write code for the different implementations and test loops. We wanted to test many conditions, scenarios and models, and found ourselves re-writing code and/or writing endless scripts. There was no software framework in which neural networks can be optimized with minimal number of lines, and in different scenarios. So, we went and created it. After writing the DNNOP framework (available for free online), we experienced an unprecedented acceleration in our arduous research efforts.

Fourth and final lesson is to get unstuck once you are stuck. After completing the first paper, we wanted to work on Robot Motion using ERL. At that time, we were still using typical deep RL settings, i.e. gradient-based optimization. For months and weeks, the robot could not learn. Now, we could either attribute this to the inherent sample complexity requirement for RL techniques, or to some limitation in our own implementation. We would save the robot's pre-programmed movements, then feed it to the network so the network can "memorize" the control actions. The network would be able to reproduce perfectly the learned sequences, as well as perform remarkably well on predicting other

unseen trajectories. Logically, this means the network has learned the robot's dynamics and how the sensory readings should affect the joints, and vice versa. This now-trained network would then be used to control the robot, and it would fail unquestionably.

When the network would fail, and upon some pondering, we would introduce some innovation. Some tweak to the way we conducted training and/or inference. While this skill did help at times, we were stuck in an endless loop of debugging and introducing improvements. The robot would not learn. It would not explore the action space either. This is when it became apparent that a new approach was needed. If it didn't work, well, it couldn't be any worse than where we were at that moment. This is where the fourth lesson was learned. If it doesn't work, learn how to think in a more wholesome manner, and "zoom out" to consider the problem from a wider perspective.

When we "zoomed out", we noticed immediately that the lack of exploration may be something not within the way we use Deep Learning but within Deep Learning itself. And so we concluded upon further consultation of core Machine Learning and Optimization literature. The gradient-descent family of optimization algorithms are created and used based on an assumption of convexity in the overall loss hypersurface (i.e. solution domain). Furthermore, there are many instances where even globally-convex problems would be challenging to the greedy-behavior of gradient descent-based algorithms, i.e. gradient-following. For example, one can easily conjure hypersurfaces which has global convexity but riddled with deep minima and saddle points. Gradient-following would not work in those scenarios. As such, we took a decision and it paid off remarkably. It was instantly

noticeable how the new exploration-based algorithms are remarkably suitable for our purposes. For those reasons, getting unstuck is perhaps the key lesson learned.

DELIVERABLES

In this section we attempt to capture what was delivered in this dissertation as well as in the PhD journey. This is an important, capsule-size collection of information. It should aid the reader capture the essence of this scholastic endeavor. The information is organized by category. Under each category are the items which were accomplished.

A. Algorithms & Software

1. Accelerated Neuroevolution algorithm 2018
2. Multiple Search Neuroevolution algorithm 2018
3. Local Search w/ Score Decay algorithm 2019
4. Developed DNNOP software framework 2018

B. Publications & Submissions

1. Published ERL in IEEE ICDL 2017
2. Published LS in IEEE UEMCON 2019
3. Published Hyperparameter Search for LS in IEEE ICICIS 2019
4. Submitted paper to ICLR 2018
5. Submitted paper to CoRL 2018
6. Submitted paper to ICLR 2019
7. Submitted paper to IEEE CEC 2019
8. Submitted paper to IEEE GECCO 2019

C. Participation & Presentations

1. Participated in RoboCup 2016
2. Attended IEEE HKN Student Leadership Conference 2015
3. Attended HRI 2016 conference
4. Attended CoRL 2018 conference
5. Presented (poster) at IEEE ICDL 2017

6. Presented (poster) at ICRA 2018
7. Presented (oral) at A2IC 2018 conference
8. Presented (oral) at IEEE UEMCON 2019
9. Presented (poster) at 2019 Lehigh University Robot Learning workshop
10. Presenting (oral) at IEEE ICICIS 2019

D. Awards & Grants

1. Received travel grant from UVA ECE Chair's office 2015
2. Received NSF EAPSI award 2016
3. Received UVA diversity award 2018
4. Received UVA ECE Best TA award 2018
5. Received workshop travel grant from Lehigh University 2019

E. Professional Experience

1. Worked w/ RoboPAL group at NTU, Taiwan 2016
2. Worked w/ Robotics team at Nvidia, Silicon Valley 2018
3. Recommended purchases for computer lab at Rice 240 and administered it for 2 years 2017-2019
4. Visiting PhD Scholar at Lehigh University 2019

F. Academic & Scholastic Experience

1. Refereed Local High and Middle School science fair 2017
2. TA for 6 courses
3. Co-designed and co-taught "Robots & Humans" w/ prof. Dugan 2017
4. Co-taught Robot Summer Camp at UVA for High School diverse students 2018
5. Taught Computer Science to Native and Diverse Middle and High School students in South Dakota 2018
6. Lectured on several occasions in several different classes 2015-2018
7. Admitted into IEEE HKN Honor Society 2014
8. President of IEEE HKN Gamma Pi UVA chapter 2014

BIBLIOGRAPHY

[1] H. Edwards and D. Edwards, “How Tesla "shot itself in the foot” by hyper-automating Model 3 production — Quartz,” *Quartz*. [Online]. Available: https://qz.com/1261214/how-exactly-tesla-shot-itself-in-the-foot-by-trying-to-hyper-automate-its-factory/amp/?fbclid=IwAR2eir_v2yU7xL75VNXDCdg_DYddBN9FJeg1JCM_sS0M5adnPVK1LYsaCF0. [Accessed: 26-Oct-2019].

[2] F. Lambert, “Tesla reveals revolutionary new wiring architecture to help robots build upcoming cars like Model Y - Electrek,” 2019. [Online]. Available: https://electrek.co/2019/07/22/tesla-revolutionary-wiring-architecture-robots-model-y/. [Accessed: 26-Oct-2019].

[3] http://www.world-wide-gifts.com, “USA - California - Dysneyland - Asimo Robot - 4 - ASIMO - Wikipedia,” 2012. [Online]. Available: https://en.wikipedia.org/wiki/ASIMO#/media/File:USA_-_California_-_Dysneyland_-_Asimo_Robot_-_4.jpg. [Accessed: 26-Oct-2019].

[4] E. Ackerman and E. Guizzo, “Honda Using Experimental New ASIMO for Disaster Response Research,” *IEEE Spectrum*, 2015. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/humanoids/honda-using-experimental-asimo-for-disaster-research. [Accessed: 09-Oct-2019].

[5] O. Sigaud and A. Droniou, “Towards Deep Developmental Learning,” *IEEE Trans. Cogn. Dev. Syst.*, vol. 8, no. 2, pp. 99–114, Oct. 2015.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[7] A. Cangelosi, *Developmental Robotics: From Babies to Robots (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2015.

[8] D. Parisi, *Future Robots: Towards a robotic science of human beings*. John Benjamins Publishing Company, 2014.

[9] Z. Chen and B. Liu, *Lifelong Machine Learning*, Second. Morgan & Claypool, 2018.

[10] G. M. Atmeh, I. Ranatunga, D. O. Popa, K. Subbarao, F. Lewis, and P. Rowe, “Implementation of an adaptive, model free, learning controller on the Atlas robot,” in *Proceedings of the American Control Conference*, 2014, pp. 2887–2892.

[11] A. Shantia, R. Timmers, L. Schomaker, and M. Wiering, “Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment,” in *Proceedings of the International Joint Conference on Neural Networks*, 2015, vol. 2015-September.

[12] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[14] Y. LeCun *et al.*, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 396–404.

[15] H. Robbins and S. Monro, "A Stochastic Approximation Method," *Ann. Math. Stat.*, vol. 22, no. 3, pp. 400–407, Sep. 1951.

[16] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," Dec. 2012.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." 22-Dec-2014.

[18] Y. Yang, X. Li, and L. Zhang, "Task-specific pre-learning to improve the convergence of reinforcement learning based on a deep neural network," in *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2016, vol. 2016-September, pp. 2209–2214.

[19] R. Salgado, A. Prieto, F. Bellas, and R. J. Duro, "Improving extrinsically motivated developmental robots through intrinsic motivations," in *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016*, 2017, pp. 154–155.

[20] U. Martinez-Hernandez, N. F. Lepora, and T. J. Prescott, "Active haptic shape recognition by intrinsic motivation with a robot hand," in *IEEE World Haptics Conference, WHC 2015*, 2015, pp. 299–304.

[21] C. Yi, H. Min, J. Zhu, and P. Yin, "Affordance discovery based on intrinsic motivation in robots," in *2016 IEEE International Conference on Robotics and Biomimetics, ROBIO 2016*, 2016, pp. 88–93.

[22] G. Baldassarre, "What are intrinsic motivations? A biological perspective," in *2011 IEEE International Conference on Development and Learning, ICDL 2011*, 2011.

[23] J. Staugaard, Andrew C., *Robotics and Ai: An Introduction to Applied Machine Intelligence*. Prentice Hall, 1987.

[24] A. Schmitz, Y. Bansho, K. Noda, H. Iwata, T. Ogata, and S. Sugano, "Tactile object recognition using deep learning and dropout," in *IEEE-RAS International Conference on Humanoid Robots*, 2015, vol. 2015-February, pp. 1044–1050.

[25] S. Funabashi, A. Schmitz, T. Sato, S. Somlor, and S. Sugano, “Robust in-hand manipulation of variously sized and shaped objects,” in *IEEE International Conference on Intelligent Robots and Systems*, 2015, vol. 2015-December, pp. 257–263.

[26] K. Sasaki, H. Tjandra, K. Noda, K. Takahashi, and T. Ogata, “Neural network based model for visual-motor integration learning of robot’s drawing behavior: Association of a drawing motion from a drawn image,” in *IEEE International Conference on Intelligent Robots and Systems*, 2015, vol. 2015-December, pp. 2736–2741.

[27] M. Spangenberg and D. Henrich, “Symbol grounding for symbolic robot commands based on physical properties,” in *2016 IEEE International Conference on Information and Automation, IEEE ICIA 2016*, 2017, pp. 62–68.

[28] A. Giagkos, D. Lewkowicz, P. Shaw, S. Kumar, M. Lee, and Q. Shen, “Perception of Localized Features during Robotic Sensorimotor Development,” *IEEE Trans. Cogn. Dev. Syst.*, vol. 9, no. 2, pp. 127–140, Jun. 2017.

[29] A. Di Nuovo, V. M. De La Cruz, A. Cangelosi, and S. Di Nuovo, “The iCub learns numbers: An embodied cognition study,” in *Proceedings of the International Joint Conference on Neural Networks*, 2014, pp. 692–699.

[30] C. Li, Q. Shi, C. Wang, Q. Huang, and T. Fukuda, “Calibration and implementation of a novel omnidirectional vision system for robot perception,” in *2016 IEEE International Conference on Robotics and Biomimetics, ROBIO 2016*, 2016, pp. 589–594.

[31] T. K. Dao, T. S. Pan, and J. S. Pan, “A multi-objective optimal mobile robot path planning based on whale optimization algorithm,” in *International Conference on Signal Processing Proceedings, ICSP*, 2017, pp. 337–342.

[32] M. Tognon and A. Franchi, “Dynamics, control, and estimation for aerial robots tethered by cables or bars,” *IEEE Trans. Robot.*, vol. 33, no. 4, pp. 834–845, Aug. 2017.

[33] S. H. Kasaei, M. Oliveira, G. H. Lim, L. S. Lopes, and A. M. Tomé, “An interactive open-ended learning approach for 3D object recognition,” in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2014*, 2014, pp. 47–52.

[34] X. Li and M. Sridharan, “Move and the robot will learn: Vision-based autonomous learning of object models,” in *2013 16th International Conference on Advanced Robotics, ICAR 2013*, 2013.

[35] M. Rucinski, A. Cangelosi, and T. Belpaeme, “An Embodied Developmental

Robotic Model of Interactions between Numbers and Space," in *Expanding the Space of Cognitive Science: Proceedings of the 23rd Annual Meeting of the Cognitive Science Society*, 2011.

[36] O. Mubin, J. Henderson, and C. Bartneck, "You just do not understand me! Speech Recognition in Human Robot Interaction," in *IEEE RO-MAN 2014 - 23rd IEEE International Symposium on Robot and Human Interactive Communication: Human-Robot Co-Existence: Adaptive Interfaces and Systems for Daily Life, Therapy, Assistance and Socially Engaging Interactions*, 2014, pp. 637–642.

[37] Q. Ren and P. Bigras, "Discrete-Time parallel robot motion control using adaptive neuro-fuzzy inference system based on improved subtractive clustering," in *2016 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2016*, 2016, pp. 1000–1006.

[38] W. Yu, S. Wang, K. Madani, and H. Feng, "A framework of autonomous knowledge transfer for robot navigation task," in *2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing: Can We Realize Awareness via Ubi-Media?, iCAST 2013 and UMEDIA 2013*, 2013, pp. 52–57.

[39] S. Contreras and F. De La Rosa, "Using Deep Learning for Exploration and Recognition of Objects Based on Images," in *Proceedings - 13th Latin American Robotics Symposium and 4th Brazilian Symposium on Robotics, LARS/SBR 2016*, 2016, pp. 1–6.

[40] L. Beyer, A. Hermans, and B. Leibe, "DROW: Real-Time Deep Learning-Based Wheelchair Detection in 2-D Range Data," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 585–592, Dec. 2016.

[41] Z. Wu, C. Valentini-Botinhao, O. Watts, and S. King, "Deep neural networks employing Multi-Task Learning and stacked bottleneck features for speech synthesis," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2015, vol. 2015-August, pp. 4460–4464.

[42] P. C. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, and T. Ogata, "Repeatable Folding Task by Humanoid Robot Worker Using Deep Learning," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 397–403, Apr. 2017.

[43] W. Chen, T. Qu, Y. Zhou, K. Weng, G. Wang, and G. Fu, "Door recognition and deep learning algorithm for visual based robot navigation," in *2014 IEEE International Conference on Robotics and Biomimetics, IEEE ROBIO 2014*, 2014, pp. 1793–1798.

[44] J. Hwang and J. Tani, "Seamless integration and coordination of cognitive skills in humanoid robots: A deep learning approach," *IEEE Trans. Cogn. Dev. Syst.*, vol. 10, no. 2, pp. 345–358, Jun. 2018.

[45] I. Guertel, G. Schillaci, and V. V. Hafner, “Using proprioceptive information for the development of robot body representations,” in *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016*, 2017, pp. 172–173.

[46] A. Di Nuovo, V. M. De La Cruz, and A. Cangelosi, “Grounding fingers, words and numbers in a cognitive developmental robot,” in *IEEE SSCI 2014 - 2014 IEEE Symposium Series on Computational Intelligence - CCMB 2014: 2014 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain, Proceedings*, 2014, pp. 9–15.

[47] D. Luo, F. Hu, W. Liu, and X. Wu, “Robot learns the concept of direction through motion activity,” in *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016*, 2017, pp. 87–94.

[48] X. Li, Y. Yang, Y. Sun, and L. Zhang, “A developmental actor-critic reinforcement learning approach for task-nonspecific robot,” in *CGNCC 2016 - 2016 IEEE Chinese Guidance, Navigation and Control Conference*, 2017, pp. 2231–2237.

[49] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, “Combining model-based policy search with online model learning for control of physical humanoids,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016, vol. 2016-June, pp. 242–248.

[50] V. C. Meola *et al.*, “Interplay of rhythmic and discrete manipulation movements during development: A policy-search reinforcement-learning robot model,” *IEEE Trans. Cogn. Dev. Syst.*, vol. 8, no. 3, pp. 152–170, Sep. 2016.

[51] Z. Kira, “Transfer of sparse coding representations and object classifiers across heterogeneous robots,” in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 2209–2215.

[52] A. M. Ghalamzan E., C. Paxton, G. D. Hager, and L. Bascetta, “An incremental approach to learning generalizable robot tasks from human demonstration,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2015, vol. 2015-June, no. June, pp. 5616–5621.

[53] G. Hettich, V. Lippi, and T. Mergner, “Human-like Sensor Fusion Mechanisms in a Postural Control Robot,” in *Proceedings of the International Congress on Neurotechnology, Electronics and Informatics*, 2013, pp. 152–160.

[54] O. Dürr, Y. Pauchard, D. Browarnik, R. Axthelm, and M. Loeser, “Deep Learning on a Raspberry Pi for Real Time Face Recognition,” *Eurographics*, no. January, pp. 0–4, 2015.

[55] M. Mori, “The Uncanny Valley: The Original Essay by Masahiro Mori,” *IEEE*

Spectrum, Jul-2012.

[56] S. Pellerano, S. Choi, and J. Rabaey, "EE2: Intelligent machines: Will the technological singularity happen?," 2017, pp. 521–521.

[57] Executive Office of the President (US), "Artificial Intelligence, Automation, and the Economy," 2016.

[58] S. Yeotikar, A. M. Parimi, and Y. V. Daseswar Rao, "Automation of end effector guidance of robotic arm for dental implantation using computer vision," in *2016 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics, DISCOVER 2016 - Proceedings*, 2016, pp. 84–89.

[59] Y. Kakiuchi *et al.*, "Development of humanoid robot system for disaster response through team NEDO-JSK's approach to DARPA Robotics Challenge Finals," in *IEEE-RAS International Conference on Humanoid Robots*, 2015, vol. 2015-December, pp. 805–810.

[60] A. Santamaria, "Teleoperated robots for live power lines maintenance (ROBTET)," in *14th International Conference and Exhibition on Electricity Distribution (CIRED 1997 - Distributing Power for the Millennium)*, 1997, vol. 1997, pp. v3-31-v3-31.

[61] R. Pringle, K. Michael, and M. G. Michael, "Unintended consequences: The paradox of technological potential," *IEEE Potentials*, vol. 35, no. 5, pp. 7–10, Sep. 2016.

[62] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, and H. V. Rios-Figueroa, "Detecting falling people by autonomous service robots: A ROS module integration approach," in *2017 International Conference on Electronics, Communications and Computers, CONIELECOMP 2017*, 2017.

[63] J. Lee, H. Takehashi, C. Nagai, and G. Obinata, "Design of a therapeutic robot for interacting with autistic children through interpersonal touch," in *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, 2012, pp. 712–717.

[64] G. Lakatos, "Dogs as Behavior Models for Companion Robots: How Can Human-Dog Interactions Assist Social Robotics?," *IEEE Trans. Cogn. Dev. Syst.*, vol. 9, no. 3, pp. 234–240, Sep. 2017.

[65] M. N. Kiyani and M. U. M. Khan, "A prototype of search and rescue robot," in *2016 2nd International Conference on Robotics and Artificial Intelligence, ICRAI 2016*, 2016, pp. 208–213.

[66] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June-2015, pp. 1–9.

[67] M. Abadi *et al.*, “TensorFlow: A System for Large-scale Machine Learning,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[68] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Sep. 2014.

[69] A. Bendale and T. Boult, “Towards Open Set Deep Networks,” Nov. 2015.

[70] J. S. D. R. G. A. F. Redmon, “(YOLO) You Only Look Once,” *Cvpr*, 2016.

[71] M. F. Stoelen, D. Marocco, A. Cangelosi, F. Bonsignorio, and C. Balaguer, “Predictive Hebbian association of time-delayed inputs with actions in a developmental robot platform,” in *Proceedings of the International Joint Conference on Neural Networks*, 2014, pp. 700–707.

[72] K. Mochizuki, S. Nishide, H. G. Okuno, and T. Ogata, “Developmental human-robot imitation learning of drawing with a neuro dynamical system,” in *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, 2013, pp. 2336–2341.

[73] Z. Wang, G. Xu, and F. Chao, “Integration of Brain-like neural network and infancy behaviors for robotic pointing,” in *Proceedings - 2014 International Conference on Information Science, Electronics and Electrical Engineering, ISEEE 2014*, 2014, vol. 3, pp. 1613–1618.

[74] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” *arXiv Prepr. arXiv1703.03864*, 2017.

[75] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” *arXiv Prepr. arXiv1712.06567*, 2017.

[76] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *Ann. Math. Stat.*, vol. 22, pp. 400–407, 1951.

[77] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[78] WERBOS and P., “Beyond Regression : New Tools for Prediction and Analysis in the Behavior Science,” *Unpubl. Dr. Diss. Harvard Univ.*, 1974.

[79] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv Prepr. arXiv1508.04025*, 2015.

[80] S. Khadka and K. Tumer, “Evolution-Guided Policy Gradient in Reinforcement

Learning," May 2018.

[81] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[82] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[83] K. O. S. and R. Miikkulainen, "Evolving Neural Networks Through Augmenting Topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[84] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl, "Automatic Feature Selection in Neuroevolution," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, 2005, pp. 1225–1232.

[85] G. J. G. Lahr *et al.*, "Adjustable interaction control using genetic algorithm for enhanced coupled dynamics in tool-part contact," in *IEEE International Conference on Intelligent Robots and Systems*, 2017, vol. 2017-September, pp. 1630–1635.

[86] V. Batchu, "Neuro-Evolution with Flappy Bird (Genetic Evolution on Neural Networks) - Threads @ IIIT Hyderabad - Quora," 2017. [Online]. Available: https://www.quora.com/q/threadsiiithyderabad/Neuro-Evolution-with-Flappy-Bird-Genetic-Evolution-on-Neural-Networks. [Accessed: 15-Oct-2019].

[87] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," Dec. 2015.

[88] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J. Rob. Res.*, vol. 34, no. 4–5, pp. 705–724, Apr. 2015.

[89] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv Prepr. arXiv1409.0473*, 2014.

[90] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *cs.xu.edu*.

[91] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Feb. 2015.

[92] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," Aug. 2015.

[93] K. Sörensen, "Metaheuristics-the metaphor exposed," *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, Jan. 2015.

[94] E. Alba, Ed., *Parallel Metaheuristics*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005.

[95] E. Ronald and M. Schoenauer, "Genetic lander: An experiment in accurate neuro-

genetic control," in *Parallel Problem Solving from Nature — PPSN III*, Springer, Berlin, Heidelberg, 1994, pp. 452–461.

[96] I. J. Goodfellow *et al.*, "Generative Adversarial Networks," Jun. 2014.

[97] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 512–519.

[98] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," Dec. 2013.

[99] S. Ruder, "An overview of gradient descent optimization algorithms." 15-Sep-2016.

[100] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." 10-Dec-2015.

[101] Z. Michalewicz, "Evolution strategies and other methods," in *Genetic Algorithms+ Data Structures= Evolution Programs*, Springer, 1996, pp. 159–177.

[102] A. Aly and J. B. Dugan, "Experiential Robot Learning with Accelerated Neuroevolution," Aug. 2018.

[103] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[104] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *NIPS-W*, 2017.

[105] T. Tusar, D. Brockhoff, N. Hansen, and A. Auger, "COCO: The Bi-objective Black Box Optimization Benchmarking (bbob-biobj) Test Suite," Apr. 2016.

[106] A. Garrett, "inspyred: Bio-inspired Algorithms in Python," *GitHub Repository*. Jun-2017.

[107] T. Schaul *et al.*, "PyBrain," *J. Mach. Learn. Res.*, vol. 11, pp. 743–746, 2010.

[108] Sonja Surjanovic; Derek Bingham, "Optimization Test Functions and Datasets," *Simon Fraser University*, 2013. [Online]. Available: http://www.sfu.ca/~ssurjano/optimization.html. [Accessed: 29-Dec-2018].

[109] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, Springer, 2011, pp. 760–766.

[110] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, 2009.

[111] P. J. M. Van Laarhoven and E. H. L. Aarts, "Simulated annealing," in *Simulated*

annealing: Theory and applications, Springer, 1987, pp. 7–15.

[112] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Fitness expectation maximization," in *International Conference on Parallel Problem Solving from Nature*, 2008, pp. 337–346.

[113] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[114] Gaortizg, "File:Bukin function 6.pdf - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/File:Bukin_function_6.pdf. [Accessed: 06-Jan-2019].

[115] Gaortizg, "File:Easom function.pdf - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/File:Easom_function.pdf. [Accessed: 06-Jan-2019].

[116] Gaortizg, "File:Eggholder function.pdf - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/File:Eggholder_function.pdf. [Accessed: 06-Jan-2019].

[117] D. Balduzzi, H. Vanchinathan, and J. Buhmann, "Kickback cuts Backprop's red-tape: Biologically plausible credit assignment in neural networks," Nov. 2014.

[118] M. Jaderberg *et al.*, "Decoupled Neural Interfaces using Synthetic Gradients," Aug. 2016.

[119] R. Battiti and G. Tecchiolli, "Training neural nets with the reactive tabu search," *IEEE Trans. Neural Networks*, vol. 6, no. 5, pp. 1185–1200, 1995.

[120] K. Hirasawa, K. Togo, J. Murata, M. Ohbayashi, N. Shao, and J. Hu, "A new random search method for neural networks learning-random search with variable search length (RasVal)," in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, vol. 2, pp. 1602–1607.

[121] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv Prepr. arXiv1312.5602*, 2013.

[122] A. Aly, G. Guadagni, and J. B. Dugan, "Derivative-Free Optimization of Neural Networks using Local Search," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, 2019, p. in press.

[123] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[124] P. L. Neary, "Automatic hyperparameter tuning in deep convolutional neural

networks using asynchronous reinforcement learning," in *Proceedings - 2018 IEEE International Conference on Cognitive Computing, ICCC 2018 - Part of the 2018 IEEE World Congress on Services*, 2018, pp. 73–77.

[125] M. P. Ranjit, G. Ganapathy, K. Sridhar, and V. Arumugham, "Efficient Deep Learning Hyperparameter Tuning Using Cloud Infrastructure: Intelligent Distributed Hyperparameter Tuning with Bayesian Optimization in the Cloud," 2019, pp. 520–522.

[126] H.-J. Yoon, J. Gounley, S. Gao, M. Alawad, A. Ramanathan, and G. Tourassi, "Model-based Hyperparameter Optimization of Convolutional Neural Networks for Information Extraction from Cancer Pathology Reports on HPC," 2019, pp. 1–4.

[127] D. Choi, H. Cho, and W. Rhee, "On the Difficulty of DNN Hyperparameter Optimization Using Learning Curve Prediction," in *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, 2019, vol. 2018-Octob, pp. 651–656.

[128] C. Liu *et al.*, "Progressive Neural Architecture Search," Dec. 2017.

[129] M. Wistuba, A. Rawat, and T. Pedapati, "A Survey on Neural Architecture Search," May 2019.

[130] M. L. Pasini, J. Yin, Y. W. Li, and M. Eisenbach, "A greedy constructive algorithm for the optimization of neural network architectures," Sep. 2019.

[131] G. Brockman *et al.*, "OpenAI Gym," Jun. 2016.

[132] OpenAI *et al.*, "Solving Rubik's Cube with a Robot Hand," Oct. 2019.