TJTMI Volume 1, Issue 1 (Spring 2022), pp. 67–89

Imitation Learning: the Machine Learning Version of Discipleship

Dr. Monica B. Vroman
PhD, Rutgers University
Researcher at Rutgers University
Visiting Research Professor of Computer Science at The Master's University

Abstract: Learning has been at the forefront of human activity for thousands of years. Machine learning, a much newer field, has recently gained popularity and gotten much attention in the media and in conversations around us. In this article, I define machine learning, describe its different branches, and explain how it works. I also answer some of the concerns and questions that people have about the future of technology, especially as it relates to advances in machine learning.

Introduction[1]

Learning, the acquisition of knowledge and skill, has been at the forefront of human activity for thousands of years. In His common grace and kindness, God has created mankind in His own image and endowed him with a special ability to acquire and process information. This intelligence is one feature that sets men and women far above the rest of creation: consider the vast amount of learning required to even read and understand this article!

One outworking of image-bearing human intelligence and creativity is technology. An area of technology that has gained increasing popularity over the last decades is artificial intelligence (AI) and particularly the offshoot of machine learning (ML). Today, there is a plethora of adaptive robots and machines that change, learn, and adjust to their environments. God has made mankind in His own image and we are making machines in our own.

[1] This introduction was coauthored with David Vroman, MDiv.

The advent of AI and machine learning has opened a new frontier of human knowledge and brought with it a multitude of important questions: In what sense can a machine learn? How does machine learning work? Should we be concerned or fearful about artificial intelligence and machine learning?

Unfortunately, in the minds of many, cinema has had more influence than science. Hollywood has painted vivid nightmares of cold and calculating tyrant robots ruling the world. According to these portrayals, humans will be surpassed by machines in every way, including intelligence, and we will be left at the mercy of merciless machines. But one need not be captivated by these doomsday scenarios to wonder about the dangers of artificial intelligence and where it might lead. In fact, the increasing prevalence of AI and machine learning coupled with rapid advances in the field do warrant careful thought.

For the Christian seeking to evaluate and submit every thought to the Truth (John 17:17, Proverbs 30:5, 2 Corinthians 10:5), thoughts about AI should be no exception. Our appraisal of AI and its potential must adhere to reality, and our responses to it ought to be shaped by the wisdom of the Bible. It is beyond the scope of this article to address spiritual responses to the realities and possibilities of AI or to address the entire breadth of AI. My goal in writing this article is to introduce the readers to the fascinating world of machine learning and to help them align their thinking with the present reality of ML. In this article, I will give the reader an idea of what machine learning is, how it works, and what are its current limitations.

This article will focus on a subset of ML called *imitation learning*. One of the most common and effective modes of learning is learning *by example*. Much of what is learned over the course of a lifetime is learned by observation and imitation. This reality can be seen in the practice of apprenticeship that has been common in many trades throughout history. It can also be seen in the pages of the Bible. The apostle Paul said, "Be imitators of me, just as I also am of Christ" (1 Corinthians 11:1, NASB). Jesus Himself regularly called on people to follow Him, to become His disciples, and to, themselves, make disciples (Matthew 28:19, 20).

In the rest of this article, we will define and examine how machines or computers are able to "gain knowledge" and in what sense they are able to learn.

Machine Learning

There is a common belief that machines, in general, are competent, exact, efficient, even infallible to a certain extent, and this view has been extended to machine learning as well. One day, my elementary school aged son proudly told me that he had successfully "fooled" his video game. This game uses the computer camera to assess how well children are able to copy drawings using game pieces of various shapes and sizes. He was missing a piece and was able to "trick" the program by making the required shape using other pieces than the ones prescribed. Another time, the same program did not accept a perfect copy of the drawing. Again, the reaction was shock. Knowing how hard it is for computers to recognize objects from pictures, the subject of an area of research called *computer vision*, I had the opposite reaction. I was pleasantly surprised at how well the computer actually had done and that it only made those two mistakes. It struck me that my son and others see computer programs as an intellectual authority. But are they? Machines are very good at accomplishing certain tasks. Aside from their immense contribution to industry, we cannot imagine some of our daily tasks such as web search and using productivity software, without them. But they do have limitations. There are certain tasks, such as computer vision or recognizing various objects in a picture, that are extremely easy for humans and very hard for computers.

We will start by defining machine learning and what a machine is. A machine is "a mechanically, electrically, or electronically operated device for performing a task; [...] a computer" [MWb]. The "machine" in machine learning is a computer or a collection of computers. In other words, machine learning is a field concerned with how computers can be programmed to learn from the data they are given. For the remainder of this article, the term *machine* will refer to a computer or a computer program.

Since God has given humans the ability to learn so that they would know Him and enjoy Him forever (according to the *Westminster Catechism* [KRM86]) and since computers cannot know God, in what sense are computers able to learn and to what purpose? Going back to the definition of learning [MWa], we ought to ask, what are the processes involved in computers gaining knowledge? How do computers study, practice, and how are they taught? Can they have experiences? If yes, what are they and how do they learn from them?

Before we talk about computer learning, we need to understand how computers do what they do. Most of us use computers on a daily basis. They assist us with the work that we do, facilitate communication, help us stay organized, and help keep us entertained. They are useful to us.

Most people know that computers need to be programmed. Programming is a process that involves writing instructions using a syntax that depends on the programming language used. Like the variety of human languages, there is also a variety of programming languages, and they go by various names: Python, C++, Java, Julia, and so on. Writing code is similar to writing instructions in another language, except that language is very close to English and shares some of the same words with English. For example, if we want to program the computer to ask the user for two numbers and then display the number that is greater in the Python programming language, which I am choosing here because it uses a simpler syntax compared to other programming languages, we would write the code in Figure 1.

The words in dark red and italicised are words that the computer does not process and, therefore, we do not have to worry about whether the computer understands them or not. When we tell the computer `print(text)` the computer takes the text given between the parentheses and shows it on the screen. It does not peek at the text to see if it can understand

```
a = input("Please enter a number, a:  \n")
b = input("Please enter another number, b:  \n")

if b > a:
    print("b is greater than a")
else:
      print("a is greater than or equal to b")
```

Figure 1: Code for printing the larger of two numbers in Python.

it. The words in **bold blue** font in the above example are special words called *instructions*. These are words that the computer recognizes and knows how to obey or follow. Other words that the computer recognizes are called *keywords* and they are underlined in the code shown above. When the computer reads an instruction or a keyword, it has a predefined set of steps it executes to follow or accomplish that instruction. The human equivalent of a programming instruction is asking a human to do something. If we ask our roommate to "please get bread on the way home," our roommate will follow certain steps to accomplish the task. Those steps might be to get in the car, drive to the store, park the car, get the wallet, get out of the car, lock the car, walk to the store, go to the bread aisle, take bread, pay for the bread, walk out of the store to the car, unlock the car, put the bread in the back seat, drive home. Thankfully, we do not have to describe all these steps to a human. We can just say something short such as "please get bread on your way home" and they bring bread when they come home. Similarly, it is helpful that we can tell the computer `print("b is greater than a")` and it shows the text `"b is greater than a"` on the screen. We do not have to tell the computer which of its switches need to be on or off or in which succession to do the steps it does to display something on its screen. Most of the computer instructions and keywords are in English and they are, typically, easy to recognize, at least to the average programmer. However, you may notice that the code I show above would be slightly awkward to read by a human and that is because it is not meant to be read by humans. Instead, it is meant to be a compromise between something that a human can communicate with the least amount of effort and something that a computer is able to understand. The awkwardness of programming languages comes from some of the fundamental differences between humans and machines, which the example of programming beautifully illustrates. On the other hand, machine learning bridges some of the gap between machines and humans. Understanding this gap, I believe, will help us appreciate some of what machine learning contributes to the field of computer science.

One of my favorite examples showing the difference in communication patterns between computers and humans is the *peanut butter and jelly activity* from CS unplugged [csu]. Like the other CS unplugged activities, the peanut butter and jelly activity does not involve the use of computers. Imagine I asked you to give me the instructions for making a peanut butter and jelly sandwich and I faithfully wrote them down. Then, I got all the necessary ingredients and utensils and faithfully followed your instructions to a tee without any flexibility. Maybe you think that this activity would successfully result in a yummy peanut butter and jelly sandwich on the first try. You may be thinking, “What could possibly go wrong?” In practice, however, this example shows that, unlike programming languages, human language is not a strict and rigid tool. Rather, it is ambiguous, redundant, full of allegories, and analogies. Human language is dependent upon references to information that is common knowledge to humans such as “lemons are sour” or “animals do not drive cars.” Small children learn this "common knowledge," but it would be extremely difficult if not impossible to encode in the “mind” of a computer [Dar20, IOM+21].

I highly recommend doing this activity with family or friends as a fun and entertaining game or looking up videos of people, typically teachers, who have done this activity in their classrooms. What this activity illustrates is that computers cannot handle ambiguity, figures of speech, exaggerations, sarcasm, etc. They take everything we tell them literally. Because of these features, programming languages are very formal, rigid, and every word has only one meaning. Programming a computer requires a certain level of skill that takes time and effort to learn. Websites such as CS Unplugged [csu], Hour of Code [hou], and others, make computer science ideas accessible to those who are interested in the big picture without the technical burden of learning programming.

While computer programming remains the main tool for getting computers to do what we want them to do, machine learning is takes a different approach: it does not involve feeding the computer the series of instructions that, when followed, lead to the computer accomplishing a certain task. Rather, machine learning algorithms describe the steps involved in how to *learn* to do the task from data. The qualitative difference between these two approaches is like the difference between following a recipe to make dinner, the equivalent of the classical programming paradigm, that is, following a set of instructions to accomplish a specific task, and learning how to drive by copying an expert driver. In the case of learning how to drive, we are learning the basics of how a car works and what its controls are, but then have the bulk of the learning come from just driving the car, seeing how the car works, and, at least in the beginning, getting feedback from the driving instructor. We could say that, in the driving example, the bulk of the learning uses the data we gather as we drive the car under various conditions and in various situations. By the time our drivers’ ed course is done, most of us are trained enough to know how to safely react to new situations based on the principles that we have learned under the supervision of a teacher. In a similar way, machine learning algorithms are programmed to learn from data or “experiences.” The task that they learn is not broken up into basic instructions that the computer can understand and know how to execute. Rather, the programming instructions make it so that the algorithm gets shaped by the data.

Definitions of Machine Learning. In 1959, well-known computer scientist Arthur Samuel, one of the first people who used the phrase "machine learning," wrote a paper describing a machine learning algorithm that learned to play the checkers game based on the experiences of playing checkers against humans and against a copy of itself [Sam59]. His program was "told" what legal and illegal moves looked like and given a few basic rules needed to play correctly. Then, it learned winning strategies from the experience of playing the game against an opponent. Arthur Samuel defined machine learning as: "the field of study that gives computers the ability to learn without being explicitly programmed."

In 1997, computer science professor Tom Mitchell wrote a book to introduce his graduate students to machine learning and give them the foundational knowledge they needed to do graduate-level research. In this book, he says, "Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience." One way to think about this definition is to imagine a computer program that is learning how to play a game from the experience of playing against a copy of itself. You can think of this copy as the copy of the instructions that make up the program or a copy of the knowledge that the program has accumulated about the game since it started playing it. Examples of such information could be, what moves are best and what moves should be avoided in various situations or how many points the program receives when making various choices in the game. The *performance measure* tells us how well the program plays the game. One such measure is the percentage of games that the program has won. When we design a machine learning task, we need to define a few parameters, such as how will we measure the success of learning the task, how will we know that the task has been learned, what choices does the algorithm have, and what information will we give the machine learning program as it makes choices. In the checkers game, the available choices are the legal moves of the various pieces; the information available to the checkers player could be how many kings the player has and how many kings the opponent has or how many pieces it has left and how many pieces its opponent has left, or a combination of those and other information.

Just as a person gets better at a skill the more they practice that skill, so machine learning algorithms get better at what they do based on experience or on the data that is available to them. The idea of improvement with experience is at the core of machine learning and this same idea is what makes machine learning threatening because, if these algorithms can improve, how much can they improve? Is there a limit to their improvement or will they, one day, take over? To understand if machine learning is a threat to humanity, we need to understand a few basics about how these algorithms learn. Machine learning is a field that is as old as computers (for example, Arthur Samuel's research on the game of checkers dates back to the late 1950s [Sam59]), but was never able to flourish to its full potential before this past decade because it was missing its main fuel, data. Machine learning is dependent on data. So, in the following, I will explain how computers learn from data.

Models. The concept of a *model* is central to machine learning. We can think of a model as a template, or a form that needs to be filled out. When we fill out a form, that form comes with some of its content already written. What we need to do is to fill in the blank information to complete the content of the form. Similarly, the structure of the model is determined ahead of time and the algorithms "fill in the blanks," as it were, using the data they are given. Said more formally, a model is a function from a set of inputs to a set of outputs, and, moreover, that function is defined by certain numbers called *parameters*. Finding the right parameters is akin to filling out a form. I will give examples of models in the next few paragraphs.

There are four main branches of machine learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

Supervised Learning. Supervised learning methods are the most widely used. Some famous examples of supervised learning algorithms are neural networks, decision trees, support vector machines, and logistic regression. These algorithms learn from labeled data. Say we wanted to train an algorithm to recognize various fruits from pictures using supervised learning and that our data would be pictures of various fruits, each picture with a label telling the algorithm the name of the fruit (Figure 2). Supervised learning has two stages: training and prediction.

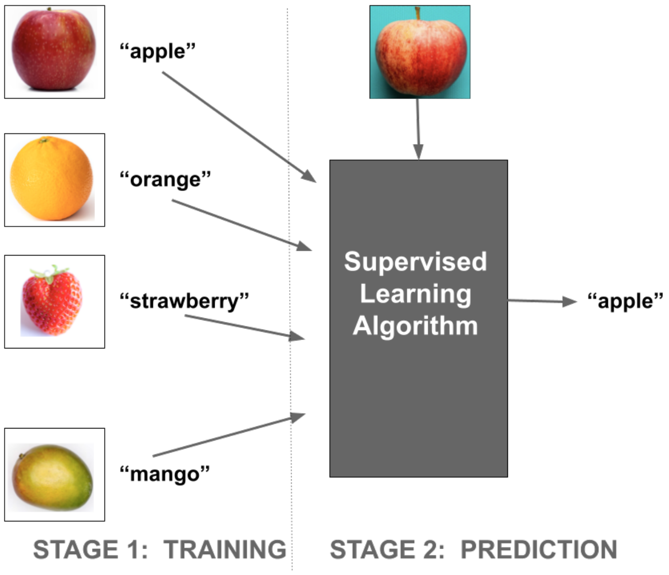

Figure 2: An overview of the supervised learning process.

During training, the algorithm learns the parameters of the model from labeled images of fruits. The goal of the training stage is that the algorithm would label a new image, which is the *prediction* or *application* stage in Figure 2.

As mentioned before, during training, a model is learned from the data. This process builds a *mapping* or *relationship* between images of fruits and words representing the names of those fruits (Figure 3). Most of the research in machine learning is focused on how this relationship is built. Imagine that the algorithm was trained using the data shown in Figure 3. One way to train the algorithm is to build a relationship between the colors in the images given in the training data and the labels of the images, a relationship that matches the training data. Imagine that the algorithm started by looking for the colors orange and green. If both were found, the algorithm would learn the label "mango" because the only fruit that is both orange and green in the training data is the mango. If the algorithm found orange and did not find green in the image, then the learned label would be "orange." If the algorithm found red and green, the learned label would be "strawberry." If red was found and not green, the learned label would be "apple." It is easy to see that this method is not very useful because some pictures of oranges may also contain green if the picture shows a leaf; this algorithm would not be able to label kiwis since no fruit in the training data is both green and brown; apples are not just red, but they can also be other colors. But this method can be a starting point in understanding how the relationship between images and labels may be built. If training is successful, the learned relationship between images and words can accurately identify the name of a fruit in a new image (Figure 2).

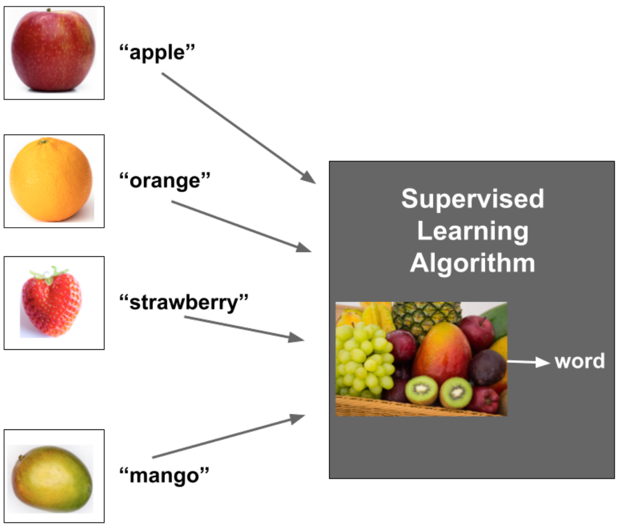

Figure 3: The training stage of supervised learning.

Every supervised learning algorithm uses labeled data, but we are not always looking to predict a label, as in the example in Figure 2. In general, the data contains some information that we are trying to predict. This information is called the *dependent variables*. In the case of labeling images of fruits, the dependent variable is the name of the fruit. The rest of the information in the data are the *independent variables*, which is the information given by the images in the example in Figure 2, for example, the pixels in the images. Labels are a special case of dependent variables.

The choice of the model or the template used usually depends on what we already know about the data. If we know that the relationship between the dependent and the independent variables is linear, that is, that when one grows, the other one grows at a similar rate, we could use a *linear model*. An example of data that could be modeled using a linear relationship is the connection between weight and height for a group of people. A linear model assumes that there is a known set of features for each training example (such as the height of a person) and that the relationship between these features and what we are trying to predict (the weight of a person) can be modeled by a line (Figure 4).

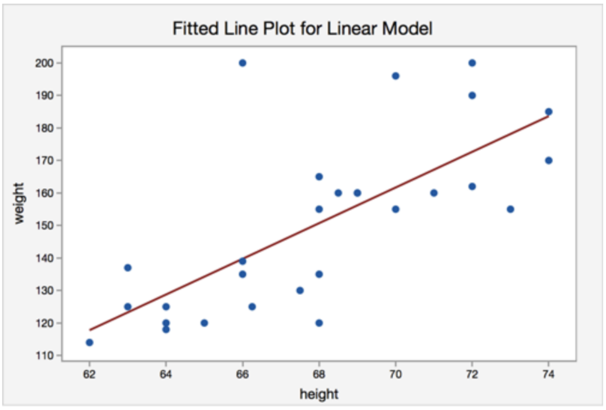

Figure 4: A plot of weight and height and the line that best models the relationship between the two [DoS21].

Limitations of Supervised Learning. Out of all the categories of machine learning, supervised learning has the greatest potential to advance the field and yield impressive results. Because of competition, companies using machine learning in their software do not make their methods public, so we do not know what data they have or what algorithms they use. Because of our limited knowledge of the best and most successful algorithms used today, it is difficult to gauge their limitations. What we do know is that the knowledge that these algorithms have is limited by the data used to train them. Google and Facebook only know about you what you "tell" them: they know what you search for, what you watch, the ads you see and those you click on, etc. They cannot learn from the information they do not have.[2]

Unsupervised Learning. In unsupervised learning, we are not given the labels of the data. We may be given pictures of various fruits and told to extract patterns from those pictures. Imagine that you have never seen an orange, apple, or strawberry. If you saw pictures of these fruits, you would not be able to name them or describe their taste or smell. You may not even know their purpose and assume that they were not edible. This is what data looks like to unsupervised learning algorithms. There are no labels to learn. All that unsupervised learning can do is to discover patterns in the data. If we had never seen and tasted certain fruits, we would not be able to describe their taste or smell or know which ones make a tasty pie, but we would be able to sort them by color or shape. Some of the most popular unsupervised learning algorithms are called *clustering algorithms* and they group similar objects together, similarly to what I just described.

Limitations of Unsupervised Learning. Unsupervised algorithms are very limited because the data they receive is unlabeled, so it is very challenging to even gauge what has been learned and how useful it is.

Semi-supervised Learning. Semi-supervised learning is a combination of supervised and unsupervised learning. During its training stage, it uses a small amount of labeled data and a large amount of unlabeled data. Because it uses some labeled data, the performance of semi-supervised learning is, in general, better than that of unsupervised learning. Also, because of the small amount of labeled data it requires, semi-supervised approaches can learn from data that lacks labels by having a human label a small amount of the training data, a process that is less time-consuming than labeling the whole data set, which would be required in a supervised approach.

Reinforcement Learning. In reinforcement learning, the program that does the learning is called an *agent*. Unlike supervised learning, there are no inputs and outputs that the agent can use to learn the model. Instead, the agent receives a *reward signal*, that is, a positive reward for desirable actions and a negative reward (or cost) for undesirable ones. The agent is programmed to choose its actions in a way that makes its total reward as high as possible. Historically, reinforcement learning has been inspired by animal learning theories [AM06]. You can think

[2] Sometimes things are a little more complicated than that. Some algorithms group users together based on various similarities and assume one piece of information on one member of a group is also true of the other members of the group. For example, *recommendation systems*, such as Netflix, group users together based on the kinds of movies they like. Say user A and user B are in the same group, they both like comedies and documentaries. Then, if A watches a movie that B has not watched and gives it a high rating, the system will recommend the movie to B.

of training a reinforcement learning agent in a similar way that a dolphin is trained, by giving it positive rewards every time it does something desirable.

The model used in reinforcement learning is called a *Markov Decision Process* (MDP) and it is used by the agent to learn how the world works. You can think of an MDP as a collection of objects. One of these objects is the collection of the *states* of the world, the equivalent of all the situations that the agent could find itself in. If the agent is learning how to park a car, the states may describe its distance from the various objects around it, its speed, etc. Another object in the MDP is the set of *actions* that are available to the agent. An agent who learns how to park a car would be able to choose to drive forward or in reverse, break, or turn the steering wheel. Another object in the MDP is a *reward function*, which tells the agent what is the reward signal it receives in each state, such as a high reward if the car is correctly parked and a negative reward or a cost if the car is too close to another car. *Transition probabilities* show the likelihood of the agent moving from one state to the next by taking actions. In a very windy environment of on an icy road, steering the car may take us in the direction we want to go with a high probability, or it may take us in a slightly different direction with a lower probability. The *discount factor* makes future rewards less appealing than present rewards, which "motivates" the agent to accomplish the task sooner rather than later. Formally, we would write that the MDP is a set:

$$MDP : (S, A, T, R, \gamma)$$

where S is the set of states, A is the set of actions, T are the transition probabilities, R is the reward function, and γ is the discount factor.

The reinforcement learning agent uses an algorithm called *value iteration* to learn the value of each of its actions in every possible state. Imagine the small world shown in Figure 5. A smiley face shows the location where the agent is; in this case, the agent is in location 8. The

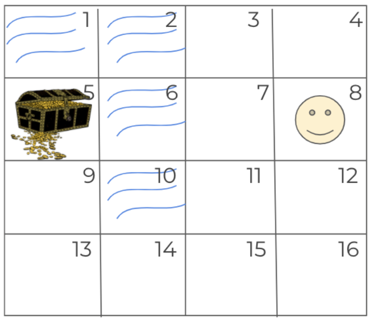

Figure 5: An example of an environment for reinforcement learning.

states have been numbered and correspond to locations in this small "grid world." The agent can move left, right, up, and down, and, every time it does an action, it receives a reward.

Imagine that the agent is in location 8 and takes the action to move to the left twice. These actions will take the agent to location 6, which is a puddle. In a puddle, suppose the agent receives a negative reward of -10 points. This reward signal is designed to discourage the agent from going into puddles. From location 6, suppose the agent takes the action to move to the left again. This action takes the agent to location 5 where, let's say, the agent receives a reward of 1000 points. This example illustrates how the agent is rewarded as it acts in the world. By convention in reinforcement learning, we assume that the agent does not know ahead of time how much reward it will receive in each location, so an initial random strategy or choice of actions makes sense. With every action, the agent learns how much reward it will receive by taking that action. After exploring the world by taking various actions in various locations, the agent learns what actions lead to ideal paths, that is paths that, when followed, yield the highest possible reward. For example, the value of location 8 can be calculated as the highest possible reward that the agent can get by following any path starting in location 8. After exploring the world enough, the agent learns that going through puddles has a cost and that the treasure chest corresponds to a high reward. The ideal path from location 8 is the one that maximizes the agent's total reward. This path avoids puddles and ends up at location 5. An example of such a path is one that goes through locations 8, 7, 11, 15, 14, 13, 9, and 5. The value of location 8, therefore, is 1000, because that is the highest possible reward that the agent can receive by choosing its actions optimally from location 8.[3]

Limitations of Reinforcement Learning. Reinforcement learning algorithms have access to limited data. They get information about the environment they are learning in and are dependent on a reward signal. They are typically applied in specialized settings and the possibility of transferring their knowledge to new settings is very limited.[4]

In reinforcement learning, the agent is given the reward function and can learn an optimal strategy, that is, the choice of actions that maximizes its total reward. In imitation learning, the agent is not given the reward function. Instead, it estimates it by observing the behavior of another agent, considered an expert at the task to be learned and assumed to behave optimally in accomplishing that task. The learner's reward function is a model that gets filled out as the learner observes the behavior of the expert. For this reason, imitation learning is also called *inverse reinforcement learning*. We will dig deeper into this process in the next section.

[3] In this calculation, I used a discount factor of 1, which is the same as saying that a future reward of 10 is considered to have the same value as a present reward of 10.

[4] To get an intuition on how reinforcement learning works and how long learning takes, watch this video on YouTube: "AI Learns to Park - Deep Reinforcement Learning" [Arz].

Imitation Learning: the Machine Learning Version of Discipleship

Let us consider an example that motivates my research in imitation learning. Suppose we want to design a robot that follows instructions given in plain English. Before the learning starts, assume the robot does not know the meaning of any English words. Instead, the system only knows the grammar (or the structure) of the instructions it will receive. For example, if we want to design a cleaning robot that receives instructions such as "Clean the garage," the robot will know that the first word in the instruction is a verb or action and the rest of the sentence is the direct object, which is the object on which the action will be performed. It is important to note that the agent will start off not knowing what the words "clean" and "garage" mean, rather it will learn the meaning of words from examples of sentences paired with the behavior demonstrating what it looks like to follow those instructions. For example, we would tell the system: "Clean the garage" and demonstrate how to clean the garage. The system will watch us clean the garage, imitate our behavior, and learn what words in the sentence "Clean the garage" mean by assigning meaning to those words based on the task that was accomplished. We show a simplified example in Figure 6. The agent demonstrating the tasks is shown as a gray diamond. Red objects are also textured with horizontal lines, yellow objects are textured with lines that go up and down, and cyan objects are textured with oblique lines.

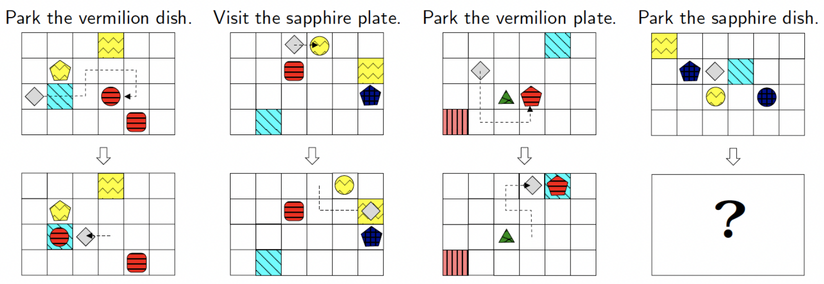

Figure 6: Examples of instructions and their corresponding desirable behavior.

In Figure 6, the first three columns are *training instances*. Each training instance is an instruction followed by an example of behavior. The behavior is shown as what the world looks like before and after the instruction is carried out or, what we call the *beginning* and *end* states. As mentioned before, the learning agent has been given enough information to know that "park" and "visit" are verbs or actions, "vermilion" and "sapphire" are adjectives, and "dish" and "plate" are nouns. The agent also has access to the world shown here as a grid world with colored and textured squares and other objects, where it can perform various actions such as moving up, down, right, and left, and learn their consequences, such as moving the agent shown as the gray diamond shape, up, down, right, and left. Some grid cells are colored and textured, which does not impact the movement of the agent, but some contain blocks, which are other objects with shapes, colors, and textures. If the agent attempts to move into a cell containing a block, that block moves in the same direction as the agent unless that block is impeded by another object or wall. In that case, no movement (of either agent or block) takes place. Using what it learns from the training instances, we would like our system to be able to receive the instruction in the fourth column, "Park the sapphire dish," and carry it out using primitive actions in the domain from the given starting point.

Consider how one might solve this problem. Looking at the first training instance, we see that the agent moved to the red circle with horizontal lines and then pushed it over to the cyan cell with oblique lines next to the yellow pentagon with lines that go up and down. It is not clear from this one training instance what "park" means, but it very likely involves some combination of the red circle, the cyan cell, and/or the yellow pentagon, based on the agent's proximity to these objects at the end of the action sequence. Looking at the third training instance, we see that the agent moved to the red pentagon and pushed it into the cyan cell. Generalizing across these two examples, it is reasonable to assume that "park" means "push to the cyan cell," since this object appears in both end states near the end location, and the argument of "park" is a description of the object to be pushed. So, the "vermilion dish" is the red circle with horizontal lines in the first example and the "vermilion plate" is the red pentagon with horizontal lines in the third. Thus, "vermilion" refers to "red with horizontal lines," "dish" to "circle," and "plate" to "pentagon." Next, we can look at the second example and see that the agent pushed the yellow circle with lines that go up and down out of the way and then proceeded to the yellow cell with lines that go up and down next to the blue pentagon. Because we have determined that "plate" refers to the pentagon, it would seem that the complement of "visit" describes an object to which the agent should become adjacent. Because the pentagon is blue, we can infer that "sapphire" means "blue." Putting the pieces together, the command to "Park the sapphire dish" can be seen to mean "push the blue circle onto the cyan cell," which can easily be carried out with a sequence of eight primitive actions.

Even though it may be easy or, at least, intuitive to conceptualize how humans such as small children could learn what language means from such examples, how would machines learn? If we use imitation learning as an example, the agent learner assumes that there is a reward function corresponding to each task that is illustrated by these examples. In other words, the behavior is generated by an agent teacher (the gray diamond in Figure 6) who is acting to maximize its reward according to its reward function. The learner does not know or have access to this reward function. Instead, it has access to demonstrations of the task to be learned. As mentioned before, this reward function is the model or template that the agent fills out as it observes the behavior of the expert modeling how the task is done. The template could be a certain type of function, for example, a linear combination of the features in the grid world. These features could be the color of the squares that the agent traverses, how long the path is that the agent takes, etc., and a linear combination of them is an expression of the following form:

$$R(s) = w_1 \cdot \text{cyan}(s) + w_2 \cdot \text{adjacentToBluePentagon}(s) + \ldots$$

where s is a state representing something akin to a location on the grid, and cyan(s) and adjacentToBluePentagon(s) are numbers that can be interpreted as the amount of information that is already in the form or template. For example, cyan(s) could be the number 1 if the location s has the color cyan and 0 if the location s does not have the color cyan. The learner is given this information already and does not have to learn it. What the agent needs to fill out using the expert's demonstrations are the "weights" of the features, or, the so-called reward function parameters, w_1, w_2, etc. These are simply numbers that reflect how important each feature is to the expert.

As the agent observes the expert, and as it assumes that the expert performs the task in an optimal way, the agent learner can fill out this template by assigning a weight to each of these features according to how important they seem to be to the expert. For example, if the expert seems to be taking the shortest path to the goal, we can assume that each step costs the expert something. Maybe the expert is factoring in the amount of battery it is using to accomplish the task and trying to make it as small as possible, such that the expert prefers shortest paths to the goal. If the expert seems to be going out of its way to step on red squares, we can assume that it gets rewarded for each red square that it steps on, for example, if red squares are places where it can charge its battery. If this is the behavior we are seeing, we can, in our template, give a high weight to red squares based on the behavior of the expert.

Maximum Likelihood Imitation Learning

It turns out that the task of filling out the model for the reward function is not straightforward. For any given behavior, one possible model is that the expert is always taking random actions. This expert does not care about any of the information in its environment; it gets the same reward no matter where it is and what it does. Even if it looks like the expert is trying to take the shortest path to the goal, or go out of its way to step on red squares, since we do not know anything about the expert's reward function, it is always a possibility that the expert's strategy is to randomly choose its actions at each step and so, it happened by chance that this expert made the choices we are observing. Because of this possibility, the imitation learning task is said to be *ill-posed*. In other words, we do not have enough information to find the expert's reward function from demonstrations of corresponding optimal behavior. The way to solve this problem is to add information that can help us arrive at the expert's reward function. This solution is accomplished by adding some assumptions to restrict the number of possible reward functions that are explanations of the expert's behavior. These assumptions vary among the different setups used in imitation learning research. For example, we could ask that the learner visits each state or location of the world about as often as the expert does. Or, we could ask that the policy or strategy of the learner be as close as possible to the policy or strategy of the expert. We can estimate the strategy of the expert by looking at the actions that the expert chooses in each state or location which we can estimate from their demonstrations. For example, we could look at how many times the expert visited a certain location during the demonstrations and count how many times the expert chose to go in each direction from that location. Then, our choice of action from that same location would follow the distribution of actions we have seen the expert choose.

Maximum likelihood inverse reinforcement learning (MLIRL), which was the focus of my graduate research [Vro14], estimates the expert's reward function by adjusting the reward function parameters such that the likelihood of the expert's demonstrations is as high as possible. As mentioned before, to solve the IRL problem, we need to make some extra assumptions. For example, we can assume that the expert's reward function is a combination of known features. Take the grid world with puddles example. The features can be: ground, puddle, start, and goal (Figure 5). The expert's reward function can be assumed to be a linear combination of these features, for example,

$$R(s) = -0.1 \cdot \text{ground}(s) + (-10) \cdot \text{puddle}(s) + (-0.5) \cdot \text{start}(s) + 1000 \cdot \text{goal}(s) \qquad (10)$$

where s is any state, and each feature ground(s), puddle(s), start(s), and goal(s) is true (or 1) if s is a location with just ground, a puddle, a starting location, and a goal location, respectively, or false (or 0) if s is not the location of ground, a puddle, the start, or a goal. For example, for location 5, the feature ground(s_5) is 1, puddle(s_5) is 0, start(s_5) is 0, and goal(s_5) is 1; the reward for s_5 is

$$R(s_5) = -0.1 \cdot \text{ground}(s_5) + (-10) \cdot \text{puddle}(s_5) + (-0.5) \cdot \text{start}(s_5) + 1000 \cdot \text{goal}(s_5) = 999.9.$$

If one was looking to interpret this reward function, it quantifies that the agent gets a high reward for reaching the goal. The agent also pays a small cost every time it takes a step toward that goal when it walks on the ground, and a higher cost when it walks in a puddle. To put it another way, the reward function parameters weigh each feature according to how much reward the agent gets in a state where the feature is true or how much it costs the agent to encounter that feature. Having a small cost for each step motivates the agent to take shorter paths to the goal.

In imitation learning we do not know the reward function, R. Instead, we know the features of each state, whether it is a ground, a puddle, a start, or a goal. What we do not know are the parameters that weigh each feature, the w_1, w_2, etc. We also know how the expert behaves and assume it is acting optimally with respect to its reward function. In other words, at each step, we assume the expert chooses the action that maximizes its total reward. The goal of imitation learning is to find an estimate of the reward function that explains the behavior we are seeing from the expert. MLIRL finds the parameters of the reward function that maximize the likelihood of the observed behavior.

More specifically, MLIRL uses an algorithm called *gradient ascent* to find the reward function parameters (see also Figure 7). To understand gradient ascent, imagine trying to find the highest point in a given area, without being able to see around you farther than one step ahead. You start in an arbitrary location on a hill or on a mountain covered with fog. Your strategy is to look around you and choose to walk in the direction of the steepest incline. In other words, at each step, choose the direction that gets you as high as possible from this point in one step. This strategy is not guaranteed to take you to the peak of the mountain, but it is guaranteed to take you to a local peak, that is, a point that is higher than any of the points around it. The gradient ascent algorithm can get stuck in local maxima (the highest point in a certain neighborhood), instead of leading to the overall, general highest point. One solution to this problem is to repeat the algorithm and choose various starting points. This strategy could lead to the global peak.

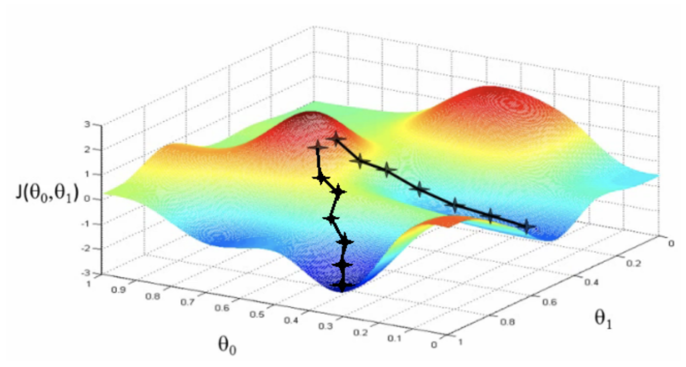

Figure 7: Gradient ascent illustration from [Coa21].

The analogy of climbing a mountain works well in the case of two features and two parameters. For example, imagine a reward function $R(s) = w_1 \cdot \text{puddle}(s) + w_2 \cdot \text{ground}(s)$. For each pair of values for the two parameters (w_1 and w_2), we can determine the value of $R(s)$ since we know the features $\text{puddle}(s)$ and $\text{ground}(s)$ for each state or grid location, s. If we have $R(s)$, we can calculate an estimate of the expert's policy and how likely the observed behavior is under that policy. These quantities are mathematical functions that depend on the values of the parameters w_1 and w_2. For example, for a given pair of values for w_1 and w_2, there is a function $L(w_1, w_2)$ that computes the *likelihood* of the observed expert behavior if $R(s) = w_1 \cdot \text{puddle}(s) + w_2 \cdot \text{ground}(s)$, since $R(s)$ is used to compute the policy, and the policy is used to compute the likelihood of the observed expert behavior. $L(w_1, w_2)$ is like the altitude (or height) of the point given by latitude w_1 and longitude w_2. When we are at the point given by latitude w_1 and longitude w_2, we simply take the step of the steepest ascent and change the latitude w_1 and longitude w_1 accordingly. For reward functions with more than two features, the same principle and method apply, but it is harder to imagine the map in more than two dimensions.

This is how the MLIRL algorithm works:

1. Start at an arbitrary "location" (that is, choose arbitrary values for the reward function parameters, w_1, w_2, ...).
2. A set of values for w_1, w_2, ... gives us a reward function. This reward function is our current estimate of the expert's reward function, $R(s)$.
3. From the estimate of R(s), we can compute an estimate of the expert's policy and the likelihood of the expert's behavior under that policy, $L(w_1, w_2, \ldots)$.
4. Take one step in the direction of the steepest ascent of L. Change w_1, w_2, ... accordingly.
5. Repeat from step 2 until the parameters w_1, w_2, ... stop changing.

When the algorithm stops, we have reached a local peak. If desired, we can repeat the algorithm using a different starting "location" for the reward function parameters, that is, different starting values for w_1, w_2, etc. in Step 1, and see if we arrive at the same peak or at a different one. If we find various peaks, we choose the one that makes L as high as possible. When MLIRL is done, it found the parameters, and, therefore, the reward function, that maximizes the likelihood of the observed expert behavior.

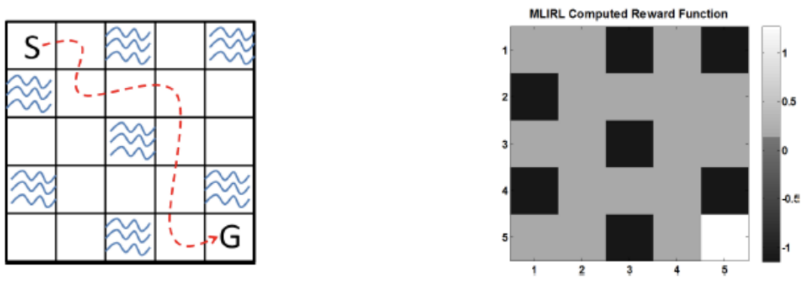

Figure 8: The reward function learned by MLIRL (on the right) using the demonstrated expert behavior (shown on the left) [Vro14].

Figure 8 shows, on the right, the reward function computed by MLIRL if the expert demonstrates the path shown on the left. This reward function assigns the highest reward to the location in the grid where the trajectory ends. The ground locations have a small positive reward of around 0.5 and the puddles have a negative reward of -1. It is interesting to note that the path shown is also the shortest path from S to G. It is possible that the expert did not care about puddles and it just so happened that, on the shortest path to the goal, it chose a path with no puddles. I have mentioned this ambiguity at the beginning of this section when discussing that the imitation learning problem is "ill-posed." MLIRL shows this ambiguity by choosing the reward function that makes the demonstrations most likely. In other words, the reward function shown on the right of Figure 8 makes the path shown on the left more likely than a reward function that would assign the same amount of reward to puddles and ground.

MLIRL Application in Carrying Out Instructions. Now, I will go back to the example I used to motivate imitation learning, that is, the example of learning how to carry out instructions in English from commands paired with demonstrations and, at the same time, learning the meaning of the words in the commands. As I mentioned when I introduced this problem, the commands are assumed to have a fixed form. In the example I showed in Figure 6, the grammar of the commands is: *verb adjective noun*. In other words, the first word is a verb in the imperative form, then comes the adjective describing the object, and, lastly, the name of the object. With this information, we could assume that each verb labels a certain reward function. In other words, there is a reward function associated with the verb "park," another one associated with the word "visit," etc. This link makes sense, since, as we have seen in the example, "park *object*" means "push object to the cyan square" and "visit *object*" means go to a square adjacent to the object. A next step could be to group together all the pairs of commands and demonstrations by the verb they use. We would put together all the pairs of commands and demonstrations that use "park," all the ones that use "visit," etc. Now, we have two demonstrations from the expert that is teaching us how to "park," and one demonstration from the expert that is teaching us how to "visit." We can use MLIRL to learn the reward function for "park," and apply it to carry out the fourth instruction, "Park the sapphire dish."

In my previous work [Vro14], I have used a simpler grammar to learn the meaning of words from demonstrations, a *bag of words* model which is also called a *uni-gram* language model. Language models are used to predict the probability of words in a sentence. In the uni-gram language model, every word has the same likelihood of appearing in the sentence and the order of words is ignored. With these assumptions, we can compute the probability of each word given each reward function. Let's say that our training data are the ones shown in Figure 9. We have demonstrations of two tasks: moving the star object into the green room (Task 1), shown on the left side of Figure 9, and going to the green room (Task 2), shown on the right.

If we know that the instructions and demonstrations on the left side of Figure 9 correspond to one reward function and the instructions and demonstrations on the right side of Figure 9 correspond to a second reward function, we can compute the probability of seeing each word when the task is Task 1 and the probability of seeing each word when the task is Task 2, under the uni-gram language model. For example, we can compute the probability of seeing the

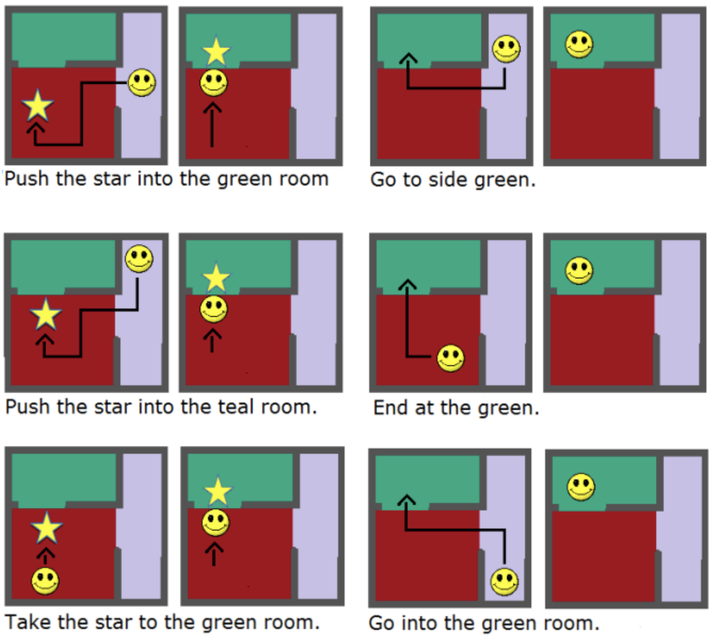
Push the star into the green room
Go to side green.
Push the star into the teal room.
End at the green.
Take the star to the green room.
Go into the green room.

Figure 9: Training data for 2 tasks: Taking the star to the green room (left) and Going to the green room (right) [Vro14].

word "push" if the instructions correspond to Task 1:

$$P(\text{"push"}|\text{Task 1}) = 2/21 \tag{11}$$

This number comes from the fact that we have three sentences instructing the agent to do Task 1, containing 21 words total. The word "push" shows up twice in these sentences, so the probability of seeing the word "push" when the demonstrated task is Task 1 is two out of 21, or 2/21.

From the demonstrations on the left in Figure 9, we can also learn the reward function for Task 1 and from the demonstrations on the right we can learn the reward function for Task 2, using MLIRL. Let's say we receive the new instruction, "Go to the green room." Which reward function should we use to follow this new command? Since we have the probabilities of each word given each task as shown in Equation (11), we can compute which task is this sentence more likely to fit. More specifically, we can compute the probabilities that this sentence would come from Task 1 and Task 2. The probability $P(\text{"Go to the green room"}|\text{Task 1})$ that this sentence would come from Task 1 is equal to

$$P(\text{"go"}|\text{Task 1}) \cdot P(\text{"to"}|\text{Task 1}) \cdot P(\text{"the"}|\text{Task 1}) \cdot P(\text{"green"}|\text{Task 1}) \cdot P(\text{"room"}|\text{Task 1}),$$

and the probability $P(\text{"Go to the green room"}|\text{Task 2})$ that this sentence would come from Task

2 is equal to

$$P(\text{"go"}|\text{Task 2}) \cdot P(\text{"to"}|\text{Task 2}) \cdot P(\text{"the"}|\text{Task 2}) \cdot P(\text{"green"}|\text{Task 2}) \cdot P(\text{"room"}|\text{Task 2}).$$

We choose the task for which this probability is higher and apply its corresponding reward function to carry out the task.

You may see that this approach is very limited because the uni-gram language model is too simple to capture the complexities of human language, even of simple instructions such as the ones shown here. However, even with such a simple language model, MLIRL can be used to figure out reward functions from demonstrated tasks. With more complex language models such as the one using the simple grammar described in the explanation of Figure 6, I believe the system would be successful in interacting with humans and carry out commands on behalf of its users. Does this mean that I have successfully designed an artificial servant of humans who watches me do my work and learns to do it for me? Not quite. This is just the beginning of an exciting line of research, with many details that still need to be figured out.

Conclusions

In this article, I gave an overview of machine learning and what it means to learn in general and what it means for machines to learn. I defined what we mean by a "machine" that, in general, it is a computer programmed to learn from the data it is given. Next, I will summarize the answers to some of the questions I raised.

What are the processes involved in computers gaining knowledge? Machine learning algorithms learn from data. They have access to the knowledge embedded in the data they are given and nothing more. Depending on the structure of the data, the approaches used can be supervised, unsupervised, semi-supervised, or reinforcement learning. Each of these categories involves different processes, but they are all based on finding patterns in the data. Supervised approaches are most widely used and they involve learning a relationship between the data and what is to be predicted. Unsupervised algorithms extract patterns in the data. In reinforcement learning, the algorithm learns a strategy based on a model of the environment and a reward signal that rewards desirable actions and punishes undesirable actions.

How do computers study, practice, and how are they taught? From the data they are given, algorithms learn relationships, patterns, or strategies, depending on the problem they are solving. While computers do not literally study or practice, they do learn from data in that they can improve their performance by using the knowledge they are given. Supervised approaches learn relationships from labeled data, unsupervised algorithms are programmed to find patterns, and reinforcement learning agents are taught through a reward signal.

Can computers have experiences? What are they and how do they learn from them? Since data is the fuel of machine learning, the data given to these algorithms can be seen as experiences. Some of these data are given to the algorithm all at once, which is a typical setup for supervised and unsupervised methods. Reinforcement learning setups are, often, closer to what we, as humans, think of an experience-based learning: they take actions in their environment, receive rewards, and change their model of the world accordingly.

How much can machine learning algorithms improve? Is there a limit to their improvement? Machine learning is limited by the data that fuels it. It is difficult to answer these questions because most advanced machine learning algorithms today are not made public, including the data they use. So, it is difficult to know how much these algorithms can improve. But we can get an idea of how advanced they are by the applications that use them. The performance of self-driving cars, handwriting and voice recognition, face and object recognition, and so on, are all tasks that are handled extremely well by artificial agents today. One obvious current limitation of artificial intelligence and machine learning is that any one agent can make a lot of progress using the data it is given in one task or in a few closely related tasks. But, unlike humans, they cannot excel at multiple tasks that are unrelated. Another obvious limitation has to do with the fact that they do not have a soul or emotions. Their minds are simulated minds and they can only improve as far as our understanding of the human mind, which is very limited.

In my personal opinion, artificial intelligence will never surpass human intelligence. As I am looking at the perfections of our Creator, His perfect wisdom, intelligence, and power, I am struck by how we, humans, as wonderful, complex, and intelligent as He has created us to be, are so far below Him. He is infinitely above us in His excellence, holiness, wisdom, and judgement. On a much smaller scale, the machines we make are limited even more in their intelligence and in what they can do. Unless we perfectly understand how our minds and intellect work, we cannot even come close to building an intelligence that mimics ours.

Acknowledgements

I gratefully acknowledge Michael Littman and others for their involvement in conceptualizing the example illustrated in Figure 6.

Bibliography

[AM06] Eduardo Alonso and E Mondragón. Associative learning for reinforcement learning: where animal learning and machine learning meet. In *Proceedings of the fifth symposium on adaptive agents and multi-agent systems*, pages 87–99, 2006.

[Arz] Samuel Arzt. AI Learns to Park - Deep Reinforcement Learnig. YouTube video: https://www.youtube.com/watch?v=VMp6pq6_QjI. Accessed on 4 January 2022.

[Coa21] Brendan Coady. Gradient ascent. *Medium.com*, 2021. Available online: https://medium.com/common-notes/gradient-ascent-e23738464a19. Accessed on 4 January 2022.

[csu] CS Unplugged. Available online: https://www.csunplugged.org/en/. Accessed on 4 January 2022.

[Dar20] Keith Darlington. Common sense knowledge, crucial for the success of AI systems. *OpenMind*, 2020. Available online: https://www.bbvaopenmind.com/en/technology/artificial-intelligence/common-sense-knowledge-crucial-for-the-success-of-ai-systems/. Accessed on 4 January 2022.

[DoS21] Penn State University Department of Statistics. Lesson 9: Linear regression foundations. *STAT 500: Applied Statsitics*, 2021. Available online: https://online.stat.psu.edu/stat500/book/export/html/478. Accessed on 4 January 2022.

[hou] Hour of Code. Available online: https://hourofcode.com/us. Accessed on 4 January 2022.

[IOM+21] Filip Ilievski, Alessandro Oltramari, Kaixin Ma, Bin Zhang, Deborah L. McGuinness, and Pedro Szekely. Dimensions of commonsense knowledge. *Knowledge-Based Systems*, 229:107347, 2021. https://doi.org/10.1016/j.knosys.2021.107347.

[KRM86] Douglas F. Kelly, Philip B. Rollinson, and Frederick T. Marsh. *The Westminster Shorter Catechism in Modern English*. Presbyterian and Reformed Pub. Co, Phillipsburg, NJ, 1986.

[MWa] Merriam-Webster.com. Learning. Available online: https://www.merriam-webster.com/dictionary/learning. Accessed on 4 January 2022.

[MWb] Merriam-Webster.com. Machine. Available online: https://www.merriam-webster.com/dictionary/machine. Accessed on 4 January 2022.

[Sam59] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959. https://doi.org/10.1147/rd.33.0210.

[Vro14] Monica C. Vroman. *Maximum likelihood inverse reinforcement learning*. Rutgers, The State University of New Jersey-New Brunswick, 2014. PhD thesis, https://doi.org/doi:10.7282/T3GQ70C8.