How to create human-level artificial intelligence

Abstract

Artificial human-level intelligence is an artificial intelligence that can accomplish tasks of a lot of value to humans. I propose that this artificial intelligence has a form of a reinforcement learning agent and the rewards are given by human (additionaly, reward comes from intristic motivation mechanism, if necessary). The reinforcement learning algorithm used must have the following characteristics among others: 1. The model architecture of the algorithm needs to be able to represent every possible policy. 2. The algorithm can represent and learn loop or recurrence. 3. The algorithm must be good at transfer learning. The order of teaching the agent tasks should be similar to the order in which human learns these tasks. This allows to enable multiple transfer learning. Multiple transfer learning is the key to teach the agent a complicated task. If this is not sufficent, then the algorithm that solves the task should be reversed engineered from the human mind at least to an extent.

Topic

This document describes a path for creating human-level artificial intelligence. Not all of the challenges leading to creating artificial human-level intelligence has been already solved (at least not by me), but this document presents a general framework for creating such intelligence. I also provide a strategy for solving the remaining challenges. The presented path is not the only path.

Definition

Human-level artificial intelligence - a computer program or a machine that serves the purpose of accomplishing tasks requiring the following skills at least: visual objects recognition, understanding of concepts from the human world, fluent communication in a natural language, analytical intelligence, theory of mind.

Human-level artificial intelligence is an artificial intelligence that is good on the tasks that have a lot of value to humans, for example: proposing solutions to global warming problem, not only on the tasks that are not very important to humans like playing chess.

Examples of tasks that human-level artificial intelligence is able to accomplish:

1. To invent an algorithm for sorting numbers and implement it in Python language.

2. To solve the logical riddle "Four Men in Hats".
3. To read a book and summarize it.
4. To have a conversation in English language.
5. To order a pizza by Internet.

The above definition is not perfectly presice but precise enough.

Plan

1. Implement proper reinforcement learning algorithm.
2. Create software for training the agent.
3. Train the agent.
4. Transfer the agent into robot. Let it learn.
5. Transfer the agent back into software.

In more detail:

1. Create a reinforcement learning algorithm that has certain characteristics (these characteristics will be described later) and implement it.
2. Create a computer program that imitates to a sufficient degree the environment in which the human intelligence exists. Put the agent into this program. The program has the interface for rewarding the agent and the reward is provided by human (the user of the program).
3. Train the agent to be able to understand and use the concepts at the base of human understanding and human deduction process, to be able to use mental strategies at the base of human problem solving, to solve some high-level intelligence tasks like solving simple mathematical problems, programming, inventing an algorithm for sorting numbers.
4. Put the agent into a machine (robot) with input and output senses similar to these that human has and transfer the model that the agent learned during its software life. Let the agent learn the human world. The robot has a remote control with the interface for rewarding the agent and the reward is provided by human (the user of the robot). The user of the robot is supposed to promise to the agent that the agent will get reward after it accomplishes a desired task. The user is supposed to give the reward after successful completion of the task. The agent is expected to accomplish the desired task (assuming that it knows how to accomplish it) as it understands that it will get reward after successful completion of the task.
5. Transfer the model that the agent learned back to its software version so that the software version can have the understanding of the things existing in the human world but not existing in the computer life.

The steps 1 and 3 should be done simultaneously because it's easier to create the proper reinforcement learning algorithm if you know which tasks are challenging for the algorithm. Therefore, creating the algorithm (step 1) and training the agent (step 3) should be done at once. Creating the software to train the agent (step 2) can be done before the steps 1 and 3 or in the middle of them, depending of the preference. Step 4 should be executed after accomplishment of steps 1, 2 and 3. Step 5 should be executed after accomplishment of step 4.

Each of the above steps will be described in greater detail in the following sections. They will be described in the following order: 2, 3, 4, 5, 1. I will describe the first step at the end because the characteristics that the algorithm must have are related to how this algorithm will be trained and used, so I need to describe how the algorithm will be trained and used before describing the characteristics.

Questions

Before I move to description of all steps in detail, I want to answer some questions that might arise after reading the above summary of the plan.

Can reinforcement learning be used to create human-level AI?

Some people might have a doubt if reinforcement learning is enough to create human-level artificial intelligence. They can say something like "Consider how a toddler learns. His grandmother can sit with him and reward him with the applause for solving a woodblock puzzle (which is an equivalent of reinforcement learning). But the vast majority of what toddler learns is learned in a different way than being rewared for something".

I propose you to look from a different perspective at it. Everything that humans do is intended to achieve happiness and avoid suffering. Obviously, sometimes we do something for the happinness of other beings, which is called altruism, but we do this because someone's happiness makes us happy. Therefore, in a sense, human intelligence is a reinforcement learning agent that is programmed to maximize its happiness and minimize its suffering and human world is a big reinforcement learning environment. Therefore, in a sense, humans learn everything that they learn through only reinforcement learning because all their actions are intended to get reward (happiness / positive emotions) and avoid negative reward (suffering / negative emotions). Therefore, if a human can learn everything that it has learned through only reinforcement learning then a proper artificial reinforcement learning algorithm can learn that too, assuming that the mechanism of human intelligence is possible to be replicated with an algorithm, assuming that the reward function of humans is possible to be replicated with an algorithm and assuming that the environment that they are in is the same. Only if the reinforcement learning algorithms/mechanisms are identical and the reward functions are identical, only then we are guaranteed that it will result in being able to learn the same things. If the reward function is different than in case of human, then we can't guarantee that the artificial reinforcement learning agent will be able to learn what humans are able to learn.

In order to make the reward function exactly the same as it is for a human intelligence, we need to replicate the human emotions (assuming that human intelligence tries to maximize its positive emotions and minimize its negative emotions). But the reward function doesn't need to be identical to human intelligence reward function, it needs to be sufficient. If the rewards are provided by human supervisor as it is proposed in this document, that might be enough or it might be necessary to implement curiosity in order to decrease the amount of human supervision that is required to train the agent. Curiosity seems to be the most important emotion motivating human learning but possibly there is also another one that is important and that will need to be implemented.

In order to create the proper reinforcement learning algorithm, we will most probably need to incorporate some unsupervised learning algorithms into it. Therefore, I'm not saying that unsupervised learning algorithms are not needed, but I argue that it is possible to create a reinforcement learning algorithm (possibly with unsupervised learning algorithm/algorithms incorporated into it) that will be able to act as a human-level artificial intelligence (assuming that the mechanism of human intelligence and human emotions can be replicated with an algorithm). My argument for that claim, as I stated above, is that human intelligence can be viewed as a reinforcement learning agent and that human intelligence is a human-level intelligence, therefore if it's possible to replicate human intelligence mechanism with an algorithm, then there exists a proper reinforcement learning algorithm that can act as a human-level artificial intelligence.

What I'm proposing in this document is to put a proper reinforcement learning algorithm into a computer program / robot and let it play the game of maximizing the return it gets in the computer program / human world. The rewards are given by human and human gives reward when the agent successfuly accomplishes the task that the human (the user of program/robot) wants it to accomplish. If needed, the reward is also provided by curiosity mechanism (rewarding surprisal) and possibly other mechanisms. Humans play the game of maximizing their happiness. Likewise, the agent will play the game of maximizing satisfaction of its human user because the rewards are given by the user and the user is supposed to give as much reward as much he is satisfied from what the agent did.

There are some safety issues related to that idea. Most importantly reward hacking. The robot might realise for example that a better way to get reward is to steal a device for giving reward and give reward on its own instead of doing what human wants them to do. But safety issues are the topic for another document, so I will not describe solutions in this document.

What about creative thinking and causal reasoning?

When reading this document, you might notice that I don't give a lot of attention to how the agent can learn creative thinking and casual reasoning despite the fact that people seem to give the most attention to that. Why? Because the agent can learn creative thinking and casual reasoning in the same way in which it can learn anything else. What is that way? This whole document explains that way (especially the sections that explain step 1 and step 3 in detail, these sections are below), but I will summarize that way in this section on an example of learning creative thinking and casual reasoning.

I assume that you have a proper reinforcement learning algorithm

Firstly, I assume that you have a proper reinforcement learning algorithm that has all of the subcharacteristics that I'll describe at the end of this document when describing step 1 and has the characteristic that it is sufficiently general (the definition of "generality" is at the beginning of the "Main characteristics" section). An algorithm that has all of these subcharacteristics and is sufficiently general is properly implemented Improved Alien, but you can use another algorithm if you have any other algorithm like that.

I assume that there exists an algorithm...

Secondly, I assume that there exists an algorithm that can solve creative thinking and casual reasoning problems. There are a few reasons to think so, among others:

1. Human intelligence is capable of creative thinking and casual reasoning and probably there exists an equivalent algorithm of the mechanism of human intelligence.
2. Algorithms that automate casual reasoning has been already invented to an extent. An example is predicting the likelihood of a fact using Bayesian network or the algorithm used in the interpreter of a Prolog language (in Prolog, you can give implications between facts and facts and the interpreter can tell you if another fact is true). These algorithms don't completely automate human casual thinking with all its complexity, but there exist an algorithm that automates casual reasoning with it's all complexity as well.
3. Algorithms that automate creative thinking has been already invented to an extent. An example are algorithms playing chess or go.

In a sense, there exists a 100% proof that there exists an algorithm that can solve all creative thinking and causal reasoning problems. The proof is as follows. The task of solving creative thinking and casual reasoning problems can be represented with the following reinforcement learning environment. The observation of the agent is a text (sequence of integers representing characters) and the agent has actions for outputing a text (actions for outputing a letter and for backspace). The agent is rewarded with 1 point, if and only if the text that it outputs is a correct solution to the problem that has been given as an input. Otherwise, the agent is not rewarded with any point. There exists an algorithm/policy that solves that task because assuming that the input text has limit of characters, then the number of possible inputs (observations) is finite. And if the number of all possible inputs of this task is finite, then there exists an algorithm with hardcoded solutions for all possible inputs. Therefore, the assumption that there exists an algorithm that can solve creative thinking and casual reasoning problems is correct (at least if the problem definition has a finite limit of characters) because there exists an algorithm that hardcodes all solutions for all possible problems. But the fact that there exists an algorithm with all hardcoded answers doesn't gives us a lot because for the machine learning algorithm to learn that algorithm/policy, it would have to experience all possible inputs in the training data which is impossible. But as I mentioned above, there are reasons to think that there are good algorithms/policies to this task that doesn't hardcode solutions for all possible problems.

Machine learning algorithms are algorithms that learn an algorithm

In order to create an artificial intelligence capable of creative thinking and casual reasoning, we possibly don't need to know that algorithm for creative thinking and casual reasoning or we don't need to know it completely. A machine learning algorithm is an algorithm that learns an algorithm. Supervised learning can be viewed as simply using an algorithm to find an algorithm based on the examples of its inputs and outputs. These examples are the training data. Based on these examples, supervised learning algorithm learns an algorithm that is later used to generate outputs for different inputs than these in the training data. The model architecture used in the algorithm, like neural network or linear regression can be viewed as a way to represent an algorithm. Some model architectures are not able to represent every possible algorithm, for example linear regression can represent only algorithms that represent a linear function.

Reinforcement learning is similar to supervised learning, but it has the concept of time and the examples are related to each other, not separated like in supervised learning. Reinforcement learning, in a sense, learns an algorithm too, but the input and output is presented in a different way than in supervised learning. If we want to create artificial intelligence capable of creative thinking and casual reasoning, then we don't necessarily have to know what's the algorithm for that because the beauty of machine learning algorithms is that they are algorithms that learn an algorithm and they can learn the algorithm by themselves.

Assuming that you have a reinforcement learning algorithm that is sufficiently general, then it will learn creative thinking and casual reasoning if you simply stimuli the reinforcement learning agent with right tasks (by definition, sufficiently general algorithm is an algorithm that can learn all of the important tasks, but not necessarily with small number of samples and small amount of computational power). The right task is for example the one mentioned above (in which the observation is text that describes the problem and the agent outputs text and is rewared when the output is a correct solution). The only problem is that the number of samples (thus also the amount of human supervision) and amount of computational power needed to learn the tasks of creative thinking and casual reasoning can be too big. And for a simple task, the reinforcement learning can simply learn a good algorithm/policy for the task. But for a complicated task like creative thinking and causal reasoning, simply applying a reinforcement learning algorithm to this task might be not enough.

Transfer learning

The first solution to this is multiple transfer learning. When a human learns a complicated task, he doesn't learn that complicated task at the beginning. He learns many simpler tasks before that. For example, if you want to teach the agent casual reasoning which is basically making conclusions from fact, you should teach him first the task of collecting the facts from experience. If you tried to teach a human casual reasoning before he learns to recognize and describe facts about experience, then you would fail. But if you teach the human recognizing the facts first and then teach him casual reasoning, then it makes learning the task much easier. The same is true for reinforcement learning agent, but the reinforcement learning algorithm needs to have the characteristic that it's good at transfer learning (that it can apply what it has learned in a different context).

Transfer learning will be described in more detail in "Transfer learning" section.

Reverse engineering the human mind

Probably there will be tasks which will be too complicated for a machine learning algorithm to learn, even after doing multiple transfer learning. The reason for this is that human mind is complicated and has a lot of mechanisms that general reinforcement learning algorithm doesn't have. For example, when doing creative thinking people often use imagination. If you took your ability to imagine things away from your mind, then you would become retarded and you wouldn't be able to solve problems. The general-purpose reinforcement learning algorithm like Improved Alien doesn't have imagination mechanism but that doesn't mean that it wouldn't be able to solve the problems that humans are using imagination for. If imagination mechanism can be represented with an algorithm, then general-purpose reinforcement learning algorithm like Improved Alien can learn that algorithm if its stumiled with the right tasks. If these tasks requires imagination to solve, then the Improved Alien will learn the algorithm for that imagination mechanism internally or it will learn some other algorithm that will be able to solve these tasks too. But the problem is that it will take incredibly high amount of samples (and thus also human supervision) and computational power for Improved Alien to learn that mechanism of imagination by itself (maybe I'm wrong and it won't). And therefore I expect that it will be necessary to reverse engineer these mechanisms from human mind and incorporate them into the general-purpose reinforcement learning algorithm (like Improved Alien). You can use the strategy for reverse engineering human mind that I described at the end of this document.

How do you know which mechanisms you need to reverse engineer? You should have a sufficiently general reinforcement learning algorithm with all the subcharacteristics described in the "Step 1: Create proper reinforcement learning algorithm" section, you should train it as if you assumed that everything will work perfectly if you do multiple transfer learning and if you run into a task that the agent can't learn for a long time, then it means that there is a mechanism that you need to reverse engineer and you need to consider why a human can learn that easily and the algorithm can't. As I said before, you can use the strategy for reverse engineering from the end of this document.

Summary of how to teach agent to be good at task X

Summarizing everything that I have written before (and after), this is summary of how you can teach the agent to be good at any task (including creative thinking and casual reasoning):

1. Have a proper reinforcement learning algorithm.
2. Use the reinforcement learning algorithm to learn the task. If this succeeded, then that's great, don't do the next steps.
3. If it's not enough, then use multiple transfer learning. I.e. divide the tasks into smaller tasks and these smaller tasks into smaller tasks and these smaller tasks into smaller tasks etc. And start from learning the smallest tasks first. You can divide it into smaller tasks as deeply as deep you know the algorithm that is the solution for this task. If it worked, then that's great, don't do next step.
4. If the task is still too complicated to learn for the reinforcement learning algorithm, then reverse engineer part of the solution to make the chunk of what the algorithm needs to learn smaller and use reinforcement learning algorithm to learn that smaller chunk. If this doesn't work, then you simply must wait longer for the reinforcement learning algorithm to learn the task because it must learn it eventually, assuming that it's sufficiently general (which is one of the characteristics of the "proper reinforcement learning algorithm").

Step 2: Create software to train the agent

Summary

Create a computer program that imitates to a sufficient degree the environment in which the human intelligence exists. Put the agent into this program. The program has the interface for rewarding the agent and the reward is provided by human (the user of the program).

Details

The program needs to imitate the human environment to a sufficient degree so that it is able to learn all of the concepts that are at the base of human understanding before we put it into a robot. For example, if the environment that the agent exists in was only a chat-like environment in which it can output letters and accepts the the text written by human as input, then it couldn't learn about visual things like for example colors. Without vision, it would be very difficult to teach it many concepts. Therefore, the environment that I propose is a 2d board on which the agent and human (the user of the program) can draw and a chat on which the agent and human (the user of the program) can communicate. The agent has actions for drawing on the board and outputting letter (text) to the chat. This environment is simple to implement but enough to teach many complicated concepts.

In this computer program, there are also buttons for giving reward to the agent so that you can reward the agent when it has successfuly accomplished the task that you are training it on.

Another thing that is necessary in this program is the "impersonation mode" (the reason for which it's necessary will be explained later). If you turn the "impersonation mode" on, then you become the agent and you can take actions that the agent can take (in other words, you are in the agent's shoes). The agent learns from these actions that you take as if they were its own, despite the fact that it doesn't choose them. This allows to decrease the time that the agent needs to learn some things greatly (I will explain why in the next section). Additionaly, the program might give the possibility to impersonate the agent's action with a different depth. I.e. if the agent has working memory of some kind, then you can choose if you want impersonate only the external actions that modify only the state of the world or if you want to impersonate also the internal actions that modify the state of the memory.

At higher stage of learning, the environment will be changed. The agent will still be able to communicate with human (the user of the program) through chat, but instead of drawing on the board, it will have the computer screen as input and will be able to use the cursor and the keyboard. This environment is sufficient to learn very complicated concepts like: the concepts at the base of the human understing, causual reasoning, programming, mathematics and many more. If it has access to the computer screen, then there are many things that it can do: it can play computer games, it can draw things in graphic editor, it can learn to read etc. This environment is sufficient to train the agent to a level at which you are clearly able to tell that you have created artificial human-level intelligence. It still won't be able to understand some concepts that doesn't exist in the computer world, like for example "chair" because chair is an object that doesn't exist on the computer screen but in external world. But it will be able to have sufficient understanding to clearly see that it can understand human world if you put it into human world. This way, you don't risk that you put into a robot something that doesn't work well enough yet to be transferred to human-level environment.

Also, it's good to also test the algorithm on games like: FrozenLake, Copy, Chess, Hamabi...

Step 3: Train the agent

Summary

Train the agent to be able to understand and use the concepts at the base of human understanding and human deduction process, to be able to use mental strategies at the base of human problem solving, to solve some high-level intelligence tasks like solving simple mathematical problems, programming, inventing an algorithm for sorting numbers.

Details

Transfer learning

Transfer learning is the key to learn a complicated task. You can notice that human intelligence uses transfer learning to learn complicated tasks. For example, at the beginning human learns to count objects and understand what number means and represents, then he reuses that knowledge to learn to compare numbers, add numbers, then he reuses that to learn multiplication of numbers, then you reuse that to learn exponentation. You don't start with learning a complicated task like programming if you haven't learned the basics before that. The reason why humans are able to learn complicated tasks like that is because human intelligence is more likely to see the patterns that are built of the patterns that it already knows. The reinforcement learning algorithm that we use must have this characteristic as well.

For the above reason, the order of teaching the agent tasks must be the same (or similar) as it is for a human. Most importantly, you don't teach the agent task X if it hasn't learned task Y yet, if X is something you wouldn't teach a human before teaching him Y and if you know that X relies on Y. For example, you wouldn't teach a human exponentation before teaching him multiplication because exponentation relies on multiplication. This is because the only way to learn a very complicated task using general-purpose reinforcement learning algorithm (that is not adjusted to be good specifically at this task) in some practical time is by using transfer learning (you can find the argumentation for that claim in the next subsection). In the context of this document, transfer learning means that before teaching the agent task X, you teach the agent the tasks that the task X relies on and that are important for being able to learn task X.

The only difference that you do comparing to the order in which humans learn is that you don't teach the agent the things that it can't understand living in the computer environment. For example, you can teach the agent complicated math before you teach them what chair is because chair doesn't exist in the agent's environment (because the agent lives inside a computer) so it can't learn what chair is at this stage. But it has everything that is needed to learn complicated math, so you can teach the agent complicated math.

Why is transfer learning necessary to learn a complicated task?

Why is it impossible to learn a very complicated task in practical time without transfer learning?

Let's suppose that the very complicated task that you want to learn is programming. This task can be represented with the following reinforcement learning environment. The observation of the agent is a text (precisely speaking, a sequence of numbers representing characters) and the agent is equipped with actions that serves the purpose of outputting a text (each action outputs one character and there is one action for backspace). The observation in this environment is always a description of a computer program (e.g. "a program that multiplies two numbers"). The agent is rewarded with 1 point if the output that it produces is the source code that will result in the described computer program after compilation of this source code. Otherwise, the reward is 0. In order to become good at this task, the agent has to learn programming, there is no other way.

What will happen if you apply a general-purpose reinforcement learning algorithm to that task? At the beginning, the agent will output random actions and do exploration like with every other tasks. But in order to learn anything, the agent has to firstly output a correct source code. If it doesn't output any correct source code, then it will never get any reward and it won't learn anything. It will take years until it will accidently produce the right source code and learns anything.

But let's suppose that we use impersonation (the feature of the training software that I have described in the previous step, impersonation is also described more precisely in "Learning through repetition (impersonation)" section below) and we can impersonate the agent in order to demonstrate the correct solution. This solves the problem that we have to wait years until the agent outputs the correct solution so that it can learn anything from it. But this is not the end, there are other obstacles that stop the agent from learning this task.

Let's suppose that we demonstrate the correct source code for different descriptions using impersonation feature. Let's suppose that we do this 10 times. Every machine learning algorithm learns based on the past samples. If we have only 10 samples, then there might be a lot of possible policies that fit into these 10 samples. There might be a lot of possible functions/algorithms that generate the right output for the given input in case of these 10 samples. And a lot of them are policies that we are not looking for and these policies will fail on the next samples. Therefore, if we want to learn a right policy, the general-purpose reinforcement learning algorithm needs a higher number of samples. For a very complicated task, the number of samples that is needed will be incredibly high, so even if you have a perfect general-purpose reinforcement learning algorithm, then it won't be able to learn that with only 10 samples (unless the algorithm is specifically adjusted to this task) because 10 samples are simply too little information. These samples need to be provided by human, therefore it's not possible to learn that in practical time because you would have to spend a lot of time supervising the agent (giving the reward). The other reason why it's impossible to learn a very complicated task without transfer learning is that even if you had the right amount of samples, then if the policy that the algorithm needs to learn is very complicated (and for the programmming task, the optimal policy is very complicated), it would take a lot of computational power to find that policy. Therefore, there are two problems with learning a complicated task: 1. It requires a lot of samples. 2. It requires a lot of computational power.

Both of these problems can be solved by transfer learning (at least to an extent). Human intelligence is able to learn programming because it's more likely to see the patterns that are build of the patterns that it has already learned. Before a human learns programming, he learns a lot of other concepts that programming is built of. If you try to teach an infant to programm, you will fail for the same reasons that you would fail with a reinforcement learning algorithm. If we want to teach a reinforcement learning algorithm a complicated task, then the algorithm must have a similar characteristic to the human intelligence - that it notices the policy that is built of parts of policy that it has already learned first, before it notices other policies ("notice a policy" in this context means: converge to a policy / find that policy / learn that policy). If the algorithm has that characteristic, then we can train the agent to learn the simpler parts firstly and then train it to learn the more complicated policy that is built of these simpler parts. This way, we can teach the agent a complicated task in a smaller number of samples despite the fact that there are a lot of other policies that fit into the past samples because the agent will be blind to see these other policies as it gives the preference to the policies that are built of the parts it has already learned. This solves the problem that in order to learn a complicated task you need a lot of samples. This also solves the problem of a big computational power needed to learn a complicated task because if the algorithm gives the preference to the policies that are built of known parts first, then it can remain blind to a lot of other policies which saves a lot of computational power that is needed to find the right policy.

More detailed transfer learning and less detailed transfer learning

Obviously, if the task is very complicated, then it might still require too many samples or too much computational power. But we can have more detailed transfer learning and less detailed transfer learning. The less detailed transfer learning is that before teaching a very complicated task X, you teach the agent 3 other tasks that leads to being able to accomplish task X. The more detailed transfer learning than this is that before teaching a very complicated task Y, you teach the agent 120 other tasks that leads to being able to accomplish task Y. In other words, more detailed transfer learning (which I also sometimes call "multiple transfer learning") is when you teach a lot of tasks before teaching the final complicated task. Less detailed transfer learning is when you teach a small number of tasks before teaching the final complicated task.

The more detailed transfer learning is, the less samples and computational power you will need in order to achieve the goal of teaching the agent a complicated task because you simply do more use out of transfer learning. Therefore, if a complicated task requires too many samples or too much computational power, then you can always split it into smaller tasks and do more detailed transfer learning which decreases the needed number of samples and the needed amount of computational power. If the smaller tasks are still too complicated (and require too many samples and computational power), then you split the smaller tasks into even smaller tasks. And you repeat that process until they are small enough to be learned with the amount of samples and computational power that you can realistically provide.

The problem is that at some point it might be difficult to split the tasks into smaller tasks, especially if the task is something that human does intuitively and doesn't know exactly how he does it. I will answer this problem in the next paragraph.

How to know how to split the task into simpler tasks

Let's suppose that you want to teach the agent to sort numbers. You know that you have proper reinforcement learning algorithm for that, but it requires too many samples or too much computational power for the agent to learn that. As stated above, the more detailed transfer learning you do, the less samples and less computational power you need. Therefore, you can use transfer learning in this case and split the task into simpler tasks that the agent will be able to learn and then learn the more complicated task of sorting the numbers. But how to know into what tasks you need to split this to?

There are two answers to that question:

1. If you want to teach the task X, then teach the agent all of the tasks that you would teach a human before teaching him X. For example, if you want to teach the agent to sort numbers, then you should teach them comparing numbers before that.
2. If you want to teach the task X, then think of what an optimal/good policy to this task would look like. Policy can be viewed as an algorithm that takes observation as an input and outputs an action (and possibly saves something into memory so that it can remember its previous observations). Think of what that algorithm would be. The algorithm can be always divided into procedures or subalgorithms (other algorithms that together and in the right orther create that algorithm). For example, if you want to teach the agent to sort numbers, then one of the algorithm for sorting numbers is finding the lowest number, writing it down (or remembering it or whatever), then finding another lowest number (out of the numbers that are left) and continuing this process until you have found all the numbers. In this case, an algorithm for finding the lowest number is a subalgorithm of that algorithm for sorting numbers. Therefore, if you want to teach the agent to sort numbers, it's good to teach them finding the lowest number first (the agent will learn the task of sorting numbers faster thanks to that, assuming that the reinforcement learning algorithm has the characteristic that it's good at transfer learning). The subalgorithm of finding the lowest number is comparing numbers. Therefore, if you want to teach the algorithm to find the lowest number, you should teach them to compare numbers before that. Another example is the task of playing Hamabi well. When a human Hamabi player chooses an action to take, he basically does two things: 1. analyzes the intentions of other players, 2. based on their analysis, chooses the best action to take. Therefore, if you had an algorithm for taking a good action when playing Hamabi, you can train him to be good at recognizing the intention before training it to be good at Hamabi because the algorithm for recognizing the intention would be a subalgorithm of most of the good algorithms for playing Hamabi.

These two methods of finding how to split the tasks into simpler tasks should usually result in the same tasks.

But what if the agent is not able to learn the task but you divided the task into simpler tasks and these simpler tasks into simpler tasks as much as you could? If you did multiple transfer learning as much as you could and the task is still too complicated to learn for the agent, then you need to help the agent to find the algorithm that solves the task by following the strategy that I have described at the end of this document. You don't need to find the entire algorithm which would be difficult many times. If you know the algorithm more or less, then you know how to split this into simpler tasks and then you can use a machine learning algorithm to learn the rest of the algorithm. In other words, if the algorithm is too complicated for the algorithm to learn and you did transfer learning, then it means that there is some mechanism in human mind that helps the human mind to accomplish this task. You need to reverse engineer that mechanism using the strategy at the bottom of the document and incorporate that mechanism/algorithm into the reinforcement learning algorithm that you use.

Different stages of learning

At different stages of life, human learns in different ways. If you want to teach someone how to play Heroes 3, then you instruct him to execute the game, then you instruct him to click in different places, you explain some things and possibly you ask some questions. But if you have a new born infant, you can't teach him this way because it doesn't understand natural language and many different things.

Stage 1: Learning through repetition (impersonation)

At the beginning of life, humans learn through repetition. One of the first thing you learn is saying the word "mum". If you think how human learns to say "mum", you can notice that they learn that through repeating their parents as they say that. If we want to do the same, then if we want to teach the agent to accomplish a task then we can have two humans, one correctly accomplishes a task and the second one rewarding the first one for accomplishing the task. This way, the agent can learn how to correctly accomplish the task. But repetition requires some complicated theory of mind (understanding that some of your actions for moving the mouth and make a sound correspond to the sound that your mom is creating) and it's easier to replace repetition with impersonation (possibly we will need to deal with that complicated theory of mind later anyway, but I'll talk about this at the end of this document when talking about reinforcement learning algorithm characteristics).

As described in the previous section, impersonation is the feature in the training software that enables you to become the agent and choose the actions for the agent so that it can treat them as their own and can learn from them. Having this feature, you don't need to have another human that demonstrates how to accomplish the task, you can simply give the agent a task (by communicating the name of the task and parameters on the chat), become the agent, accomplish the task and give the reward to the agent after succesful completion of the task. It's you who have succesfuly completed the task, but the agent will not recognize that and will learn from your actions. Sometimes you might also need to demonstrate an incorrect behaviour and in this case you don't reward the agent after that. If you repeat that process over and over again, then the agent will eventually learn that in that situation (when the name of that task is communicated on the chat), if it executes the desired behaviour, it will get the reward. Obviously, sometimes the task might be too complicated to learn quickly, but the answer how to deal with that problem is in other sections, this section is only about how to demonstrate how the task can be solved to the agent.

For example, if you want to train the agent to draw a triangle on the board in different colors (color is the parameter of the task), then you repeat the above process many times:

1. Type "Triangle([color])" on the chat (or you can simply type "draw [color] triangle" if you prefer) to communicate the task. You should substitute "[color]" with some color, for example: red, blue, green.
2. Turn the "impersonation mode" on.
3. Draw a traingle in the given color on the board to demonstrate the correct behaviour to the agent.

4. Turn the "impersonation mode" off.
5. Give reward to the agent for succesful completion of the task.

You should do this process with demonstrating incorrect behavior and not rewarding the agent as well because otherwise the agent will learn that it will get the reward regardless of what it does, instead of learning the correct behavior.

Stage 2: Learning through the exchange of information

At the later stage, when the agent is at the sufficient level of understanding, the agent can learn through the exchange of information. For example, if you want to teach it how to play heroes 3, then you simply communicate to the agent through the chat to execute heroes 3, then you instruct it to click in some places, you explain some things to it, you ask some questions. Until you achieve that stage, you need to use impersonation.

Step 4: Transfer the agent into robot. Let it learn.

Summary

Put the agent into a machine (robot) with input and output senses similar to these that human has and transfer the model that the agent learned during its software life. Let the agent learn the human world. The robot has a remote control with the interface for rewarding the agent and the reward is provided by human (the user of the robot). The user of the robot is supposed to promise to the agent that the agent will get reward after it accomplishes a desired task. The user is supposed to give the reward after successful completion of the task. The agent is expected to accomplish the desired task (assuming that it knows how to accomplish it) as it understands that it will get reward after successful completion of the task.

Details

Purpose

There are two purposes of this step:

1. We want to have artificial human-level intelligence not only in the form of a software but also in the form of a robot so that it can accomplish physical tasks like cleaning the room.
2. The software version will not understand concepts from the human world that doesn't exist in the computer life. For example, it won't understand what chair is because it doesn't exist on the computer. If you tell "create a website about chair please" to the software version, then the software agent will not be able to accomplish that without knowing what chair is. Therefore you need to transfer it to the human world, let it learn about human world and then transfer it back to the computer.

Speech2text and text2speech

In the previous step, the environment in which the agent was learning contained a chat where the agent could talk to a human. The robot should have speech2text and text2speech mechanisms. Speech2text mechanism automatically converts the speech of other humans into text which is one of the input senses of the agent. Text2Speech automatically converts the text of the agent into speech and text is one of the output senses of the agent. The agent should be transfered into robot in such way that the input sense of receiving a message through chat from the human in the training software corresponds to the text input in the robot and the output sense of typing on the chat corresponds to the text output in the robot. Thanks to that, the robot will be good at speaking and communicating from the very beginning of its robot life because it has learned that in its software life and everything will remain the same for the agent when it comes to communicating through text. The only thing that will change is that the speech of a human will be automatically converted to text (which wasn't necessary in software) and the text that the agent outputs will be automatically converted to speech (which wasn't necessary in software). But this will happen outside of the agent so everything will remain the same for the agent when it comes to communication by language.

Impersonation

The robot needs to have impersonation mode as well. I.e. if you want, you can turn the impersonation mode on and then you can control the robot so that it can learn by demonstration as I have described in the previous sections.

Usage

The robot has a remote control with the interface for rewarding the agent and the reward is given by human (the user of the robot) by pushing the buttons on the remote control. The user of the robot is supposed to communicate the task to the agent and promise to the agent that the agent will get reward after it accomplishes the desired task. The user is supposed to give the reward after successful completion of the task. The agent is expected to accomplish the desired task (assuming that it knows how to accomplish it) as it understands that it will get reward after successful completion of the task.

Step 5: Transfer the agent back into software

Summary

Transfer the model that the agent learned back to its software version so that the software version can have the understanding of the things existing in the human world but not existing in the computer life.

Details

After the agent in the robot has learned about the human world, you transfer it back to the computer life. The purpose of this is that the software version will not understand concepts from the human world that doesn't exist in the computer life. For example, it won't understand what chair is because it doesn't exist on the computer. If you tell "create a website about chair please" to the software version, then the software agent will not be able to accomplish that without knowing what chair is. Therefore you need to transfer it to the human world, let it learn about human world and then transfer it back to the computer.

Step 1: Implement proper reinforcement learning algorithm

Summary

Create a reinforcement learning algorithm that has certain characteristics (these characteristics are described below) and implement it.

Details

Main characteristics

Generality - tells how big variety of tasks the algorithm is able to learn. Generality is not about how many samples it will need or how much computational power it needs to learn the task, it cares only about if it is able to learn that at all (even if we have to wait 10^99 years for the algorithm to learn that). If I say that an algorithm is narrow, then it means that it can't learn a high variety of tasks (e.g. can play only chess). If I say that an algorithm is highly general, then it means that it can learn a high a variety of tasks. This definition is not perfectly precise but precise enough.

The characteristics that the reinforcement learning algorithm that we can use to create artificial human-level intelligence (by following the above described path) must have are as follows:

1. It must be sufficiently general (ideally, completely general).
2. It can't require too much human supervision to learn what is needed to learn (it can't require too many samples).
3. The above conditions must be met assuming that we don't use more computational power that we are able to produce.

Why these characteristics?

By definition, generality means that it is able to learn a large variety of tasks. If we want to create artificial human-level intelligence, then it needs to be able to learn tasks like natural language communication, casual reasoning, creative thinking, programming etc. All of these tasks can be represented with a reinforcement learning environment. Sufficient generality means that the algorithm is theoretically able to learn all of these tasks (but doesn't mean that it will take little time or little computational power to learn that). If it can learn these tasks at all, then the only obstacles are that it might take too many samples to learn that (meaning that it will require too much human supervision because rewards are given by human, so if there are a lot of samples, then human needs to give a lot of rewards) or that it can require too much computational power. Therefore the second characteristic is that it doesn't require too many samples that we are able to practically provide (which corresponds to the first obstacle) assuming that we follow the path that was described above and the third characteristic is that it doesn't require too much computational power (which corresponds to the second obstacle). If all of these conditions are met, then after execution of the above described path, we will get artificial human-level intelligence.

Subcharacteristics

For each of these characteristics, there are subcharacteristics. If characteristic X is a subcharacteristic of the characteristic Y, then it means that if an algorithm doesn't have the characteristic X then it won't have the characteristic Y for sure or the probability of having the characteristic Y is very low.

The reason why it's worth to introduce subcharacteristics is that it might be difficult to think how to create an algorithm that meets the main characteristics but if you split it into subcharacteristics, then it might be easier to create that algorithm.

The model architecture must be able to represent every policy

// change to the model must be able to represent every function that takes trajectory as an input and outputs an action

"The architecture of the model that the algorithm uses can represent every possible policy" is a subcharacteristic of "the algorithm must be sufficiently general".

Most reinforcement learning algorithms have two parts: the first one learns a model based on the past samples and the second one executes the best action according to the current model (except exploration, when it doesn't take necessarily the best action).

"A model M represents a policy P" means that if we put that model M into the reinforcement learning algorithm that this model is related to, then the algorithm will act according to the policy P, assuming that we turn the exploration off (so the algorithm always takes the best action according to the model) and that we turn learning part of the algorithm off so that the model is frozen and doesn't change.

An architecture of the model A can represent a policy P, if and only if there exists a model M that can be represented with the architecture A and model M represents policy P.

For example, if we have a reinforcement learning algorithm which model architecture is a linear classifier in which the input of that classifier is the observation of the agent and the output is an action that the agent chooses, then that linear classifier architecture can't represent every possible policy. An example of a policy that it can't represent is a policy that accepts an array with the shape (2,) and chooses action 1, when observation[0] xor observation[1] == 1 and chooses action 2 otherwise.

Another example, if we have a reinforcement learning algorithm which model architecture is a vanilla neural network in which the input of that neural network is the observation of the agent and the output is an action that agent chooses, then that vanilla neural network architecture can't represent every possible policy. An example of a policy that it can't represent is a policy in which the chosen action depends on a past observation. For example, a policy that outputs an action 1, if the observation_one_tour_ago[0] == 1 can't be represented by that architecture. Vanilla neural network with some kind of a memory system could learn that policy but vanilla neural network with memory counts as a diffferent architecture than vanilla neural network alone.

Why is that characteristic a subcharacteristic of "it must be sufficiently general"? Because if a model architecture that is used in the algorithm can't represent some policy, then there is likely a task where that policy is an optimal policy (or a good policy) and the algorithm can't learn that policy because it can't represent that policy. Therefore, it's very likely that there will be an important task that the algorithm can't learn if the model architecture of the algorithm can't represent every possible policy.

For example, an algorithm using linear classifier model architecure that was mentioned above can't learn a task in which you need to calculate XOR function and output the action corresponding to the result. Another example, an algorithm using vanilla neural network model can't learn a task of having a conversation on a chat which displays only the last message because in order to learn that task, it needs to remember previous moments (so that it understands the context of the message).

If an algorithm doesn't have this subcharacteristc, then it doesn't mean that there is 100% chances that the algorithm is not sufficiently general, but the chances of this algorithm being sufficiently general are very low. If an algorithm has this subcharacteristic, it doesn't mean that the algorithm is sufficiently general.

The algorithm is good enough at transfer learning

"The algorithm is good enough at transfer learning" is a subcharacteristic of "The algorithm can't require too much human supervision to learn what is needed to learn (it can't require too many samples)".

As I stated above (in the details of "Step 3: Train the agent"), transfer learning is necessary to teach a general-purpose reinforcement learning agent a complicated task (like programming for example) in practical time (with the amount of samples that we can realistically provide and with the amount of computational power that we can realistically provide). The reason why it's necessary has been explained in the section "Why is transfer learning necessary to learn a complicated task?" of this document.

Therefore, the algorithm needs to be good enough at transfer learning. Being good at transfer learning means that it notices the policy that is built of parts of policy that it has already learned first, before it notices other policies ("notice a policy" in this context means: converge to a policy / find that policy / learn that policy). In other words, if the agent has learned algorithm A, then if the algorithm A is subalgorithm of algorithm B, then it can learn B quicker if it has already learned A than if it hasn't learned A already. For example, if it has already learned a task of comparing numbers, then it should be able to learn a task of sorting numbers quicker because all sorting algorithms consists of the algorithm for comparing numbers. If it doesn't learn sorting numbers quicker when it has already learned comparing numbers, it means that the reinforcement learning algorithm tries to learn the new task from scratch instead of reusing what it has already learned. And you can't realistically learn a complicated task in practical time without reusing what you have learned in previous tasks. It should also work in the opposite direction, i.e. it should be also able to learn algorithm A quicker if it has already learned B, if A is a subalgorithm of B. It should be also able to learn algorithm C quicker if it has already learned algorithm B and algorithm C = A + D, where A is subalgorithm of B and D is some other algorithm.

Likewise with other subcharacteristics, the fact that the algorithm is good enough at transfer learning doesn't mean that it won't require too much human supervision but if it's not good enough then we can be almost sure that it will require too much human supervision.

The algorithm can represent and learn loop or recurrence

The characteristic "The algorithm can represent and learn loop or recurrence" is a subcharacteristic of "The algorithm can't require too much human supervision to learn what is needed to learn (it can't require too many samples)".

Why?

Imagine that you have a task in which you get two numbers as input and you need to output a multiplication of these two numbers (more precisely - an action that corresponds to the number that is multiplication of these two numbers). The optimal policy for that task is therefore an algorithm that multiplies two numbers. This algorithm requires using loop (for, while, repeat...) because multiplying numbers X and Y is adding the number Y, X number of times. The sum that you will get is the result of multiplication. You can also use recurrence because every loop can be replaced with recurrence and every recurrence can be replaced with loop, so they are interchangeable. Can you accomplish that task without using loop or recurrence (assuming that you can't use the operator for multiplication)? Yes, because you can hardcode all possible answers, assuming that the set of all possible inputs is finite (we can make that assumption because we will use the reinforcement learning algorithm on environments in which the input array is an array of integers and that array has some fixed shape, so the number of all possible inputs is finite). Therefore, in order for the reinforcement learning algorithm to meet the subcharacteristic that the model architecture need to represent every possible policy, the model architecture doesn't need to be able to represent loop or recurrence. But if the algorithm can't represent somehow loop or recurrence, then for some tasks it will take insane amount of samples to learn a good policy. Why? Let's take multiplication of numbers as an example. The only algorithms that can be learned with not insane number of samples (assuming that the set of all possible inputs is really big) that are the solution to this task include a loop or recurrence. Other algorithms (these which don't include loop or recurrence) that can give always correct answer, like the algorithm with hardcoded answers for all possible input, will take an insane number of samples to learn this algorithm because in order to learn that algorithm, the reinforcement learning agent must experience all possible inputs. Therefore, there are tasks for which the only optimal algorithm that can be learned with small number of samples is an algorithm that includes loop or recurrence. Therefore, if a reinforcement learning algorithm shouldn't require too much human supervision (too many samples), then the model architecture of that algorithm should be able to represent loop or recurrence. And the algorithm should be able to learn that loop or recurrence.

The algorithm has good enough computational complexity

The characteristic "The algorithm has good enough computational complexity" is a subcharacteristic of "The above conditions must be met assuming that we don't use more computational power that we are able to produce".

What does it mean that the algorithm has good enough computational complexity? It means that it needs to have good enough time complexity and good enough space complexity. What does good enough time complexity and good enough space complexity mean? Is linear complexity good enough? Is quadratic complexity good enough? Is logarithmic complexity good enough? It depends on the variable towards which it has logarithmic/linear/quadratic computational complexity. For example, if value of this variable is 10^50 on average, then linear complexity is unacceptable, logarithmic complexity should be all right. In my calculations, I usually assume that a computer is able to do about 10^9 operations in acceptable time (let's say 30 seconds). Obviously, it depends on the computer but if you have 100 times better computer than this, then it will do 10^11 operations in acceptable time, so it's usually not a big difference in terms of what time complexity is acceptable - if you have a exponential complexity towards a variable that is equal 1000 then it won't work regardless of the computer you use, if you have a linear complexity towards a variable that is equal to 1000, then it will work regardless of the computer you use. If you have a time complexity of an algorithm and you can predict more or less the value of the variable on average, then you can predict how many operations it will take more or less. If that takes more than 10^9 operations, then I assume that the time complexity is unacceptable, if it takes less than 10^9 operations, then I assume that the time complexity is acceptable. A similar approach can be applied to space complexity as well - if a space complexity is such that something requires more than 10^9 of memory cells (for example, an array lenghth is higher than

10^9), then I assume that the space complexity is unacceptable.

Which algorithm does have these characteristics?

Improved Alien has the following characteristics out of the characteristics and subcharacteristics that were mentioned above:

1. The algorithm is sufficiently general.
2. The model architecture must be able to represent every policy.
3. The algorithm is good enough at transfer learning.
4. The algorithm can represent and learn loop or recurrence
5. The algorithm has good enough computational complexity

Improved Alien lacks the following characteristics out of the characteristics and subcharacteristics that were mentioned above:

1. The algorithm doesn't require too much human supervision (requires too many samples).
2. The algorithm is not too computationally expensive.

In order to create the algorithm that will have all of the characteristics and subcharacteristics (including these two that Improved Alien lacks) that were mentioned above, this is what you need to do:

1. Have Improved Alien algorithm.
2. Implement it.
3. Train it according to the plan described in this document.
4. During training, you will run into tasks that will be challenging for the Improved Alien to learn (because it lacks these two characteristics). For example: tasks requiring visual recognition because it would take a lot of time for Improved Alien to learn visual recognition.
5. There must be some mechanism in human mind that makes it easy for human mind to learn that task and Improved Alien lacks that mechanism. For example: the mechanism of human mind is such that human mind can learn to recognize objects and Improved Alien doesn't have any mechanism for recognizing objects.
6. Reverse engineer that mechanism using the strategy from the bottom of this document. Incorporate that mechanism/algorithm into Improved Alien so that it can learn that task quickly. For example: create an algorithm for visual recognition and incorporate that algorithm into Improved Alien.
7. After you incorporate that mechanism/algorithm into Improved Alien, it will become good at the task. Return to the step 3 and repeat steps 4-7 for every task that will turn out to be challenging.

Improved Alien algorithm extended with all needed mechanisms/algorithms as described above is called Humanized Alien. This algorithm will have all characteristics and subcharacteristics that are needed to create human-level intelligence.

I expect that the mechanisms that will need to be incorporated into Improved Alien in order to create Humanized Alien are as follows:

1. Feature recognition - an algorithm that recognizes features (especially visual objects) because this is something that Improved Alien can learn in theory, but in practice it will take too many samples and computational power to learn that.
2. Long-term memory - for example Improved Alien will have problems to answer questions from the distant past because its memory works in a different way than human memory.
3. Imagination - humans use imagination in the process of creative thinking. Improved Alien doesn't have imagination and I expect that it will need to have that in order to be able to become as good and as quick at creative thinking as human is.
4. Theory of mind - human mind has some magical ability to recognize the actions of other people and knowing which of their actions correspond to what other humans do. Possibly, a machine learning will be able to learn that ability without any special help, but possibly that ability will need to be also reverse engineered.
5. Curiosity - curiosty-driven learning might possibly need to be incorporated into Improved Alien as well.

Probably there will be some other challenges that I'm not aware of as well (they will come up during the training).

I have general ideas for how to solve all of the above problems, except theory of mind.

Strategy for reverse engineering human mind