A Roadmap towards Machine Intelligence

Tomas Mikolov[1], Armand Joulin[1] and Marco Baroni[1,2]

[1]Facebook AI Research

[2]University of Trento

Abstract

The development of intelligent machines is one of the biggest unsolved challenges in computer science. In this paper, we propose some fundamental properties these machines should have, focusing in particular on *communication* and *learning*. We discuss a simple environment that could be used to incrementally teach a machine the basics of natural-language-based communication, as a prerequisite to more complex interaction with human users. We also present some conjectures on the sort of algorithms the machine should support in order to profitably learn from the environment.

1 Introduction

A machine capable of performing complex tasks without requiring laborious programming would be tremendously useful in almost any human endeavor, from performing menial jobs for us to helping the advancement of basic and applied research. Given the current availability of powerful hardware and large amounts of machine-readable data, as well as the widespread interest in sophisticated machine learning methods, the times should be ripe for the development of intelligent machines.

We think that one fundamental reasons for this is that, since "solving AI" at once seems too complex a task to be pursued all at once, the computational community has preferred to focus, in the last decades, on solving relatively narrow empirical problems that are important for specific applications, but do not address the overarching goal of developing general-purpose intelligent

machines. In this article, we propose an alternative approach: we first define the general characteristics we think intelligent machines should possess, and then we present a concrete roadmap to develop them in realistic, small steps, that are however incrementally structured in such a way that, jointly, they should lead us close to the ultimate goal of implementing a powerful AI.

We realize that our vision of artificial intelligence and how to create it is just one among many. We focus here on a plan that, we hope, will lead to genuine progress, without by this implying that there are not other valid approaches to the task.

The article is structured as follows. In Section 2 we indicate the two fundamental characteristics that we consider crucial for developing intelligence– at least the sort of intelligence we are interested in–namely *communication* and *learning*. Our goal is to build a machine that can learn new concepts through communication at a similar rate as a human with similar prior knowledge. That is, if one can easily learn how subtraction works after mastering addition, the intelligent machine, after grasping the concept of addition, should not find it difficult to learn subtraction as well.

Since, as we said, achieving the long-term goal of building an intelligent machine equipped with the desired features at once seems too difficult, we need to define intermediate targets that can lead us in the right direction. We specify such targets in terms of simplified but self-contained versions of the final machine we want to develop. Our plan is to "educate" the target machine like a child: At any time in its development, the target machine should act like a stand-alone intelligent system, albeit one that will be initially very limited in what it can do. The bulk of our proposal (Section 3) thus consists in the plan for an interactive learning environment fostering the incremental development of progressively more intelligent behavior.

Section 4 briefly discusses some of the algorithmic capabilities we think a machine should possess in order to profitably exploit the learning environment. Finally, Section 5 situates our proposal in the broader context of past and current attempts to develop intelligent machines.

2 Desiderata for an intelligent machine

Rather than attempting to formally characterize intelligence, we propose here a set of desiderata we believe to be crucial for a machine to be able to autonomously make itself helpful to humans in their endeavors. The guiding

principles we implicitly considered in formulating the desiderata are to minimize the complexity of the machine, and to maximize interpretability of its behavior by humans.

2.1 Ability to communicate

Any practical realization of an intelligent machine will have to *communicate* with us. It would be senseless to build a machine that is supposed to perform complex operations if there is no way for us to specify the aims of these operations, or to understand the output of the machine. While other communication means could be entertained, natural language is by far the easiest and most powerful communication device we possess, so it is reasonable to require an intelligent machine to be able to communicate through language. Indeed, the intelligent machine we aim for could be seen as a computer that can be programmed through natural language communication, or as the interface between natural language and a traditional programming language. Importantly, humans have encoded a very large portion of their knowledge into natural language (ranging from mathematics treatises to cooking books), so a system mastering natural language will have access to most of the knowledge humans have assembled over the course of their history.

Communication is, by its very nature, *interactive*: the possibility to hold a conversation is crucial both to gather new information (asking for explanation, clarification, instructions, feedback, etc.) and to optimize its transmission (compare a good lecture or studying with a group of peers to reading a book alone). Our learning environment will thus emphasize the interactive nature of communication.

We argue that natural language can also channel, to a certain extent, non-linguistic communication. For example, in the simulation we discuss below, a Teacher uses natural language to teach the Learner (the intelligent machine being trained) a more limited and explicit language (not unlike a simple programming language) in which the Learner can issue instructions to its environment through the same communication channels it uses to interact with the teacher. Similarly, the intelligent machine can be instructed to browse the Internet by issuing commands in the appropriate code through its usual communication channels, mastering in this way a powerful tool to interact with the world at large. Language can also serve as an interface to perceptual components, and thus update the machine about its physical surroundings. For example, an object recognition system could transform

raw pixel data into object labels, allowing the machine to "see" its real-life environment through a controlled-language modality.

Under this radically language-centric approach, the machine only needs to be equipped with the linguistic I/O channels, thus being maximally simple in its interface. The machine can learn an unlimited number of new linguistic codes enabling it to interface, through its unified I/O channels, with all sorts of communicating entities (people, other machines, perceptual data encoded as described above, etc.).

Finally, while we propose language as the general *interface* to the machine, we are agnostic about the nature of the internal representations the machine must posit to deal with the challenges it faces. In particular, we are not making claims about the internal representations of the machine being based on an interpretable "language of thought" (Fodor, 1975). In other words, we are not claiming that the machine should carry out its internal reasoning in a linguistic form: only that its input and output are linguistic in nature.

The claim that the linguistic communication channels are necessary for the development of human-like intelligence seems relatively uncontroversial. We realize that our focus on the linguistic side of intelligence may limit the learning machine in the development of skills that we naturally gain by observing the world around us. There seems to be a fundamental difference between the symbolic representations of language and the continuous nature of the world around us, as we perceive it (but see the work of Max Louwerse, e.g., Louwerse, 2011, for evidence that language also encodes perceptual aspects of our knowledge). If this will turn out to be an issue, we can extend the tasks the machine is facing in the simulated environment with some that are more perception-oriented. This can be accomplished by exposing the machine to simple encodings (for example, bit streams) of continuous input signals, such as images. The related tasks can then focus on understanding first the basic properties of continues variables, and gradually extend to assignments such as identifying shapes in 2D images. Note that including such tasks would not require us to change the design of our learning framework, only to introduce novel scripts.

To give a few examples of how a communication-based intelligent machine can be useful, consider an intelligent machine helping a scientist with research. First of all, the communication-endowed machine does not need to pre-encode a large static database of facts, since it can retrieve the relevant information from the Internet. If the scientist asks a simple question such

as: *What is the density of gold?*, the machine can search the Web to answer: $19.3g/cm^3$.

Most questions will however require the machine to put together multiple sources of information. For example, one may ask: *What is the most promising direction to cure cancer, and where should I start to meaningfully contribute?* This question may be answered after the machine reads a significant number of research articles online, while keeping in mind the perspective of the person asking the question. Interaction will likely play a central role, as the best course of action for the intelligent machine might involve entering a conversation with the requester, to understand her motivation, skills, the time she is willing to spend on the topic, etc.

Going further, in order to fulfill the request above, the machine might even conduct some independent research by exploiting information available online, possibly consult with experts, and direct the budding researcher, through multiple interactive sessions, towards accomplishing her goal.

2.2 Ability to learn

Arguably, the main flaw of "good old" symbolic AI research (Haugeland, 1985) lied in the assumption that it would be possible to program an intelligent machine largely by hand. We believe it is uncontroversial that a machine supposed to be helping us in a variety of scenarios, many unforeseen by its developers, should be endowed with the capability of *learning*. A machine that does not learn cannot adapt or modify itself based on experience, as it will react in the same way to a given situation for its whole lifetime. However, if the machine makes a mistake that we want to correct, it is necessary for it to change its behavior - thus, learning is a mandatory component.

Together with learning comes *motivation*. Learning allows the machine to adapt itself to the external environment, helping it to produce outputs that maximize the function defined by its motivation. Since we want to develop machines that make themselves useful to humans, the motivation component should be directly controlled by users through the communication channel. By specifying positive and negative rewards, one may shape the behavior of the machine so that it can become useful for concrete tasks (this is very much in the spirit of reinforcement learning, see, e.g., Sutton and Barto, 1998, and discussion in Section 5 below).

Next, we will present a simulated environment designed to foster the capacity to perform reward-driven learning. In Section 4 below, we will then

come back to the discussion of what kind(s) of learning the machine must be able to accomplish in order to develop in the simulated environment at a sufficient rate, so that we may hope it could scale towards learning to perform complex tasks.

3 A simulated ecosystem to educate communication-based intelligent machines

In this section, we describe a simulated environment designed to teach language to an intelligent machine, and how to use it to learn to operate in the world. The simulated ecosystem should be seen as a "kindergarten" providing basic education to intelligent machines, and we want the machines trained in this controlled environment to later become connected to the real world to learn how to help humans with their various needs.

The ecosystem I/O channels are generated by an automatic mechanism, since having humans in the loop from the beginning would complicate and slow down both testing and development of new machine learning techniques. The environment must be challenging enough to force the machine to develop sophisticated learning strategies (essentially, it should need to "learn how to learn"). At the same time, complexity should be manageable, i.e., a human put into a similar environment should not find it unreasonably difficult to learn to communicate and act within it, even if the communication takes place in a language the human is not yet familiar with. After mastering the basic language and concepts of the simulated environment, the machine should be able to interact with and learn from human teachers. This puts several restrictions on the kind of learning the machine must come to be able to perform: most importantly, it will need to be capable to extract the correct generalizations from just a few examples, at a rate comparable to human learners.

The examples we provided above of how the intelligent machine could help a researcher to find out about promising directions in studying cancer require abstract information seeking and manipulation skills quite different from the concrete path-finding tasks we feed below to our learning machine. We believe however that, just like for humans, problem solving should first be grounded in concrete tasks, to be later extended to more abstract scenarios by analogical means (e.g., from finding a route in a maze to finding the right

information in a paper collection; Hofstadter and Sander, 2013; Lakoff and Johnson, 1999).

3.1 High-level description of the ecosystem

Agents To develop an artificial system that is able to incrementally acquire new skills through linguistic interaction, we should not look at the training data as a static set of labeled examples, as in common machine learning setups. We propose instead a dynamic ecosystem akin to that of a computer game. The Learner (the system to be trained) is an actor in this ecosystem.

The second fundamental agent in the ecosystem is the Teacher. The Teacher assigns tasks and rewards the Learner for desirable behaviour, and it also provides helpful information, both spontaneously and in response to Learner's requests. The Teacher behaviour is entirely scripted by the experimenters (once the Learner has been trained by the Teacher bot, it should be ready to continue its career amidst actual human users).

Like in classic text-based adventure games, the Environment is entirely linguistically defined, and it is explored by the Learner by giving orders, asking questions and receiving feedback (although graphics does not play an active role in our simulation, it is straightforward to visualize the 2D world in order to better track the Learner's behaviour, as we show through some examples below). The Environment is best seen as the third fundamental agent in the ecosystem. The Environment behaviour is also scripted. However, since interacting with the Environment serves the purpose of observation and navigation of the Learner surroundings ("sensorimotor experience"), the Environment uses a controlled language that, compared to that of the Teacher, is more restricted, more explicit and less ambiguous. One can thus think of the Learner as a very high-level programming language, that accepts instructions from the programmer (the Teacher) in natural language, and converts them into the machine code understood by the Environment (this is indeed a potential long-term practical application of systems trained in our ecosystem). The Teacher and Environment vocabularies partially share their form and semantics, allowing the Learner to bootstrap from one to the other. For example, if the Teacher tells to *swim through a pond* to a Learner that has never heard of swimming before, the latter might still issue the order *I swim through the pond* to the Environment, which will lead it to experience swimming for the first time. In other words, postulating partially overlapping communication codes should facilitate the acquisition of "translation"

functions enabling the Learner to turn linguistic instructions into operational plans. At the same time, since the Environment accepts only a limited set of commands, the Learner will have to translate more complex tasks given by the Teacher, such as *go forward until you hit a wall*, into a sequence of commands to the Environment.

While it is straightforward to extend and modify the simulated world by enlarging and updating the Environment vocabulary, for the sake of the examples to follow we assume this world to be split into discrete cells that the Learner can traverse horizontally and vertically. The world includes barriers, such as walls and water, and a number of objects the Learner can interact with (a yellow pear, a small blue mug, etc).

Interface channels The Learner experience is entirely defined by generic *input* and *output* channels. The Teacher, the Environment and any other language-endowed agent write to the input stream. Reward is also written to the input stream. Ambiguities are avoided by prefixing a unique string to the messages produced by each actor. The Learner writes to its output channel, and it is similarly taught to use unambiguous prefixes to address the Teacher, the Environment and any other agent or service it needs to communicate with. Having only generic input and output communication channels should facilitate the seamless addition of new interactive entities, as long as the Learner is able to learn the language they communicate in.

Reward Reward can be positive or negative (1/-1), the latter to be used to speed up instruction by steering away the Learner from dead ends, or even damaging behaviours. The Teacher, and later human users, control reward in order to train the Learner. We might also let the Environment provide feedback through hard-coded rewards, simulating natural events such as eating or getting hurt. Like in realistic biological scenarios, reward is sparse, mostly being awarded after the Learner has accomplished some task. As intelligence grows, we expect the reward to become *very* sparse, with the Learner able to elaborate complex plans that are only rewarded on successful completion, and even displaying some degree of self-motivation. Indeed, the Learner should be taught that short-term positive reward might lead to loss at a later stage (e.g., hoarding on food with poor nutrition value instead of seeking further away for better food), and that sometimes reward can be maximized by engaging in activities that in the short term provide

no benefit (learning to read might be boring and time-consuming, but it can enormously speed up problem solving–and the consequent reward accrual– by making the Learner autonomous in seeking useful information on the Internet). Going even further, during the Learner "adulthood" explicit reward could stop completely. The Learner will no longer be directly motivated to learn in new ways, but ideally the policies it has already acquired will include strategies such as curiosity (see below) that would lead it to continue to acquire new skills for its own sake.

We assume binary reward so that no sophisticated policy quantifying relative degrees of reward is needed to interact with the Learner. In this way, when the scripted Teacher is replaced by humans, they will only have to praise the Learner when it succeeds and blame it when it fails, without worrying about how much to reward it (and if they do want to control the amount of reward, they can simply reward the Learner multiple times). The Learner objective should however maximize *average reward over time*, naturally leading to different degrees of cumulative reward for different courses of action (this is analogous to the notion of expected cumulative reward in reinforcement learning, which is a possible way to formalize the concept). Even if two solutions to a task are rewarded equally on its completion, the faster strategy will be favored, as it leaves the Learner more time to accumulate further reward. This automatically ensures that efficient solutions are preferred over wasteful ones. Moreover, by measuring time independently from the number of simulation steps, e.g., using simple wall-clock time, one should penalize inefficient learners spending a long time performing offline computations.

Incremental structure In keeping with the game idea, it is useful to think of the Learner as progressing through a series of levels, where skills from earlier levels are required to succeed in later ones. The division into levels is meant for programmers to structure the simulation into banks of coherent tasks, but levels should not be exposed as explicit roadblocks in the game. It is more productive to let the Learner discover its own optimal learning path by cycling multiple times through blocks of tasks, rather than forcing it to follow a rigid difficulty-based level sequence (last but not least, because what counts as more or less difficult will partially depend on the design of specific Learners).

At the beginning, the Teacher trains the Learner to perform very sim-

ple tasks in order to kick-start linguistic communication. The teacher first rewards the Learner when the latter repeats single characters, then words, delimiters and other control strings.

In a subsequent block of tasks, the Teacher leads the Learner to develop a semantics for these symbols, by encouraging it to associate linguistic expressions with actions. This is achieved through practice sessions in which the Learner is trained to repeat strings that function as Environment commands, and it is rewarded only when it takes notice of the effect the commands have on its state. At this stage, the Learner should become able to associate linguistic strings to primitive moves and actions (*turn left*).

Next, the Teacher should assign tasks involving action sequences (*find a red apple*), and the Learner should convert them into sets of primitive commands (simple "programs"). The Teacher will, increasingly, limit itself to specify an abstract end goal (*bring back enough food for me and you*), but not recipes to accomplish it, in order to spur creative thinking on behalf of the Learner (e.g., if the Learner gets trapped somewhere while looking for food, it might discover how to build a ladder to climb a wall by stacking solid objects). In the process of learning to parse and execute higher-level commands, the Learner should also be trained to ask clarification questions to the Teacher (e.g., by initially granting reward when it spontaneously addresses the Teacher, and by the repetition-based strategy we illustrate in the examples below). With the orders becoming more general and complex, the language of the Teacher will also become richer and more ambiguous, challenging the Learner capability to handle such common natural language phenomena as polysemy, vagueness, anaphora and quantification.

To support user scenarios such as the ones we envisaged in Section 2 above and those we will discuss at the end of this section, the Teacher should eventually teach the Learner how to "read" natural text, so that the Learner, given access to the Internet, can autonomously seek for information online. Another important direction for advanced tasks will be to include other learning agents in the ecosystem, and instruct the original Learner to train them. The Learner can then discover that these agents are similar to it, developing a "theory of mind" (Whiten, 1991) in order to guide them. This could then allow the Learner to accomplish more ambitious tasks by replicating itself, and coordinating a large workforce.

Just like during child education, the Learner must first take its baby steps, in which it is carefully trained to accomplish simple tasks such as learning to construct basic commands. However, for the Learner to have any hope

to develop into a fully-functional intelligent machine, we need to aim for a "snow-balling" effect to soon take place, such that later tasks, despite being inherently more complex, will require a lot less explicit coaching, thanks to a combinatorial explosion in the background abilities the Leaner can creatively compose (like for humans, learning how to surf the Web should take less time than learning how to spell).

Time off for exploration Throughout the simulation, we foresee phases in which the Learner is free to interact with the Environment and the Teacher without a defined task. Systems should learn to exploit this time off for undirected exploration, that should in turn lead to better performance in active stages, just like, in the dead phases of a video-game, a player is more likely to try out her options than to just sit waiting for something to happen, or when arriving in a new city we'd rather go sightseeing than staying in the hotel. Since curiosity is beneficial in many situations, such behaviour should naturally lead to higher later rewards, and thus be learnable.

Evaluation Given that the learning objective is to maximize average reward in time, and the environment setup does not naturally support a distinction between a training and a test phase, the machine must carefully choose reward-maximizing actions from the very beginning. In contrast, evaluating the machine only on its final behavior would overlook the number of attempts it took to reach the solution. Such alternative evaluation would favor models which are simply able to memorize patterns observed in large amounts of training data. In many practical domains, this approach is fine, but we are interested in machines capable of learning truly general problem-solving strategies. As the tasks become incrementally more difficult, the amount of required computational resources for naive memorization-based approaches scales exponentially, so only a machine that can efficiently generalize can succeed in our environment. We will discuss the limitations of machines that rely on memorization instead of algorithmic learning further in Section 4.3 below.

We would like to foster the development of intelligent machines by employing our ecosystem in a public competition. Given what we just said, the competition would not involve distributing a static set of training/development data similar in nature to the final test set. We foresee instead a setup in which developers have access to the full pre-programmed environment for

a fixed amount of time. The Learners are then evaluated on a set of new tasks that are considerably different from the ones exposed in the development phase. Examples of how test tasks might differ from those encountered during development include the Teacher speaking a new language, a different Environment topography, new obstacles and objects with new affordances, and novel domains of endeavor (e.g., test tasks might require selling and buying things, when the Learner was not previously introduced to the rules of commerce).

3.2 Examples from early stages of the simulation

Preliminaries and notation The following examples assume that the Learner has already learned how to pay attention to the Teacher and repeat character strings, and show how the Teacher guides it from these basic skills to being able to solve relatively sophisticated Environment navigation problems by exploiting interactive communication. The examples illustrate specific instances from a larger set of trials following similar templates, that should involve a variety of objects, obstacles and possible actions. The presented examples do not aim to exhaustively cover all learning-fostering strategies that might be implemented in the ecosystem.

In the illustrative exchanges below, we present the input to the Learner (messages from Teacher and Environment, as well as Reward, prefixed by **T:**, **E:** and **R:**, respectively) in the left column, and the Learner output on the right (messages addressed to the Teacher are prefixed by **@T:**, those for the Environment are prefixed by **@E:**). We assume different I/O identification prefixes to disambiguate meta-linguistic statements (see, e.g., "**T:** give order **@E:** I move." in the first exchange below, where it's only the Teacher speaking all the way). The full stop serves as end-of-message delimiter (line breaks are used for formatting purposes, and should be treated as white space; similarly, hyphens are inserted for readability and should be ignored). We use an ellipsis (three dots) to mark a sequence of messages on any stream we do not report (e.g., the Learner exploring a number of solutions before zeroing in on the right one). Finally, we interleave the conversations with comments, shown in grey.

We only illustrate "polite" turn-taking, in which messages do not overlap, and agents start writing to the communication channels only after the end-of-message symbol has been issued. We do not however assume that interaction must be constrained in this way. On the contrary, there are advantages in

letting entities write to the communication channels whenever they want: for example, the Teacher might interrupt the Learner to prevent him to complete a command that would have disastrous consequences, or the Learner may interrupt the Teacher as soon as it figured out what to do, in order to speed up reward (a simple priority list can be defined to solve conflicts, e.g., Teacher's voice is "louder" than that of Environment, etc.).

The Learner learns to issue Environment commands We start with the Teacher teaching the Learner to produce Environment commands, first by rote repetition channeled to the right message recipient (**@E**), then by encouraging it to translate from the Teacher's own natural language to the controlled Environment language. At this stage, reward is not tied to understanding what happened (the "semantics" of the commands). Still, a good Learner might take note of the actions triggered by the commands. The following are example conversations that may happen during the learning process:

Input: *Output:*

T: give order **@E:** I move.

fdakjf13ktj53u9gs5 5 3#.

character soup without addressee, no reward

...

@T: dsfjkasjfk;aljsf.

character soup with wrong addressee, no reward

...

@E: fglk4$3wfgg.

right addressee, character soup, no reward

...

give order **@E:** I move.

E: you moved.

R: 1.

Learner moves and is rewarded, although the solution is inefficient

T: give order **@E:** I move.

...

@E: I move.

E: you moved.

R: 1.

Learner figured out minimal string leading to reward

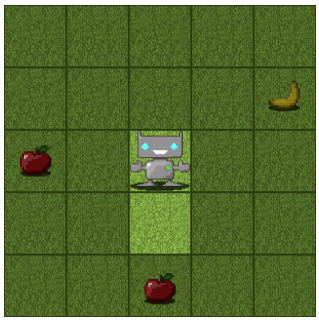
Further skills the Learner will need to develop include segmentation of Teacher's input, and production of separate commands for the Environment. At first, this may be rather trivial, as shown in this example:

Input:	*Output:*
T: give orders **@E:** I move and **@E:** I look.	
	@E: I move and **@E:** I look.
no reward, because Environment does not parse complex commands	
...	
T: give orders **@E:** I move and **@E:** I look.	
	@E: I move.
E: you moved.	
	@E: I look.
E: there is an apple.	
R: 1.	

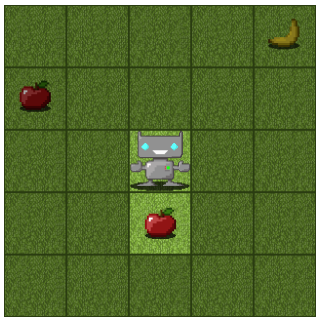
Eventually, the Learner should learn to switch fluently between Teacher and Environment communication codes, translating the Teacher's linguistic messages into motor commands to the Environment (an example of such exchange is visualized in Figure 1).

Associating language to actions Next, the Learner is encouraged to take notice of what happens in the Environment and associate the Teacher's language to states of the world, by being rewarded only when its orders result in the corresponding actions being effectively carried out in the world. An example task is illustrated in Figure 2, where the Learner will not get reward because it is no longer sufficient for it to correctly translate instructions into Environment commands: the commands must also have the intended effect on the world. In this specific case, order execution fails because the Learner encountered a barrier.

Learning to generalize By adding sufficient variety to the trials above, the Learner should start to get hold of the compositional nature of commands (*turn right* and *turn left* share systematic characteristics, e.g., they both cause a change in what the Learner sees; the Learner can pick X only when X is in the cell in front of him, etc.). The next trials help the Learner further

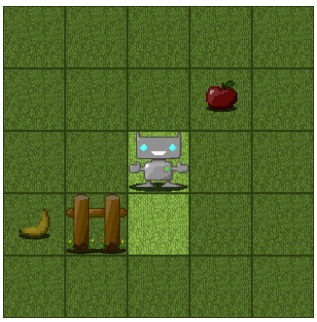


Input: *Output:*
T: move and look.
@E: I move.

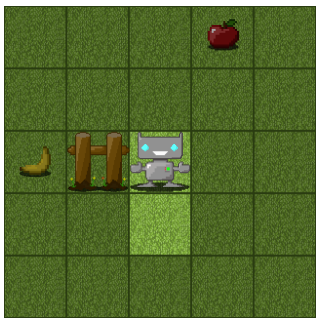


Input: *Output:*
E: you moved.
@E: I look.
E: there is an apple.
R: 1.

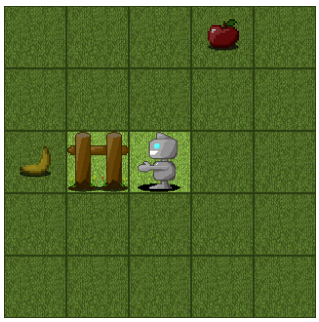
Figure 1: Example of how the simulation might be visualized to help developers track Learner's behaviour. The left panel represents the Environment at the stage in which Learner issues the move command, the right panel depicts the Environment after this command is executed. A lighter shade of green marks the cell the Learner occupies, and the one it is turned towards. These cells are directly observable. (Best viewed in color.)



Input: *Output:*
T: move, turn right and move.
@E: I move.



Input: *Output:*
E: you moved.
@E: I turn right.



Input: *Output:*
E: you turned right.
@E: I move.
E: you can't move.

Figure 2: The Teacher gives the order *move, turn right and move.* The Learner is however blocked by a barrier after it turns right, and reward is delayed, because the order is not executed. (Best viewed in color.)

move away from literal command string memorization, and generalize across categories of actions and objects (*turn left/right* are instances of *turning*, *apples* and *pears* are *objects*, etc.). This is demonstrated in the following example:

Input:	*Output:*
T: turn and move.	
	@E: I turn.
E: *(no action)*	
	...
	@E: I turn left.
E: you turned left.	
	@E: I move.
E: you can't move.	
there is a barrier in front of Learner	
	...
	@E: I turn right.
E: you turned right.	
	@E: I move.
E: you moved.	
R: 1.	

In the next example, the Learner is asked to pick to pick some object that is in front of him, without specifying what the object is:

Input:	*Output:*
T: pick an object.	
	@E: I pick an object.
E: *(no response)*	
	@E: I pick the apple.
E: *(no response)*	
	@E: I pick the pear.
E: you picked the pear.	
R: 1.	

As we just illustrated, initially the Learner will apply an exhaustive search strategy, listing all objects it knows of to find one it can pick. Later, the Teacher should teach the *look* command, and the Learner should be able to discover a faster strategy than exhaustive search:

Input:	*Output:*
T: pick an object.	
	@E: I look.
E: you see a pear.	
	@E: I pick the pear.
E: you picked the pear.	
R: 1.	

Learning new strategies of course does not imply that the Learner can safely forget the ones it has previously acquired. For example, in some tasks the *look* command might not work (because, say, it is too dark and the Learner cannot see what is in front of him). In such case, an efficient Learner should find it easy to re-use previously learned skills, such as exhaustive search.

Understanding higher-level orders The next batch of trials aims at developing the ability to decompose higher-level commands into a sequence of basic actions, first fixed (*move twice*), then flexible (*find an apple*). The general teaching strategy is to provide sets of paired trials: In the first trial, the Teacher describes the task explicitly as a sequence of primitive actions, and gives the high-level name of the order. In the second trial, Teacher issues an equivalent high-level order, and the Learner is rewarded if it goes through the same sequence as in the first trial. The Teacher could start by teaching iteration counters:

Input:	*Output:*
T: move and move.	
	@E: I move.
E: you moved.	
	@E: I move.
E: you moved.	
R: 1.	
T: this is called move two times.	
	...
T: move two times.	
	...
	@E: I move.
E: you moved.	

Input: *Output:*

@E: I move.

E: you moved.
R: 1.

. . .

T: move and move and move.

@E: I move.

E: you moved.

@E: I move.

E: you moved.

@E: I move.

E: you moved.
R: 1.
T: this is called move three times.

. . .

T: move three times.

. . .

@E: I move.

E: you moved.

@E: I move.

E: you moved.

@E: I move.

E: you moved.
R: 1.

The Learner should eventually be able to apply modifiers productively to different actions without excessive further training (e.g., apply *three times* to *turn left*).

Next, the Teacher can interactively show how to segment high-level tasks, such as *finding something*, into atomic action sequences. For example, to find an apple, one can develop a simple strategy of going forward until an apple is found. The Teacher might initiate the Learner to this new skill as follows:

Input: *Output:*
T: move and look.

@E: I move.

E: you moved.

@E: I look.

Input:	*Output:*
E: you see grass.	
T: move and look.	
	@E: I move.
E: you moved.	
	@E: I look.
E: you see an apple.	
R: 1.	
T: this is called find an apple.	

Note that one such example is not sufficient for the Learner to understand what exactly we call *find an apple*, as multiple interpretations are valid: maybe we just wanted it to execute the given commands twice. Thus, there should be multiple training sequences of variable length, to clarify that the task is really to apply an `until` loop, i.e., to iterate *move* and *look* commands until the object the Learner is searching for is found.

Further tasks can define composite skills, such as *getting an apple*, which would consist of first finding an apple, and then picking it up. Another generalization would be to include multiple objects the Learner should be searching for. Adding obstacles to the Environment would further complicate the challenge. The previous strategy of simply going forward until an object is found will not work anymore, as the Learner would stop at the first obstacle. We can either expect the Learner to independently develop better search strategies involving turning, or we can add further examples where the Teacher shows the Learner how to improve through more direct supervision.

Interactive communication Tasks such as *finding an apple* might involve a long random search. Thus, we want to kick-start interactive communication, so that the Learner can be efficiently directed by the Teacher (and eventually by humans) to the right cell. In a first set of trials, the Learner is rewarded for repeating a *how to* request uttered by the Teacher (addressing it back to the teacher via the **@T:** prefix), and following the precise instructions produced by the Teacher in response to the request:

Input:	*Output:*
T: ask me how to find an apple.	
	@E: ask me how to find an apple.
wrong addressee, wrong prefix	

Input:	*Output:*
	...
	@T: how to find an apple.
T: turn right and move and move and pick the apple.	
	@E: I turn right.
E: you turned right.	
	@E: I move.
E: you moved.	
	@E: I move.
E: you moved.	
	@E: I pick the apple.
E: you picked the apple.	
R: 1.	

Trials such as this one are later interspersed with trials where the Learner is assigned a task it can in principle accomplish by random search, but taking the initiative by issuing a *how to* request and then following the precise directions provided by the Teacher will considerably speed up reward.

Algorithmic knowledge Some tasks illustrated above require understanding basic control flow structures. For example, parsing action modifiers implies a simple form of counting, and in order to find things the Learner must implement an `until` (equivalently, `while not`) loop. Similarly, the command *get out of the grass* calls for a `while` loop. Efficient completion of more advanced tasks, e.g., *return home*, implies development of more complex algorithms, such as path-finding. After acquiring a bank of such algorithms, the Learner should be able, in advanced stages of the simulation, to productively combine them in order to succeed in full-fledged novel missions that involve accomplishing a large number of hierarchically-structured sub-goals (*find somebody who will trade two apples for a banana*).

As we discussed in Section 3.1, the Learner must essentially find out how to compose programs based on the descriptions given in natural language by the Teacher. The programs produce very simple instructions that are understood by the Environment, which can be seen as a sort of CPU. From this point of view, the intelligent agent we aim to train is a bridge between the Teacher (later to be replaced by a human operator) and a traditional computer that understands only a limited set of basic commands and needs

to be manually programmed for each single task.

3.3 Interacting with the trained intelligent machine

Finally, we provide a motivating example of how an intelligent machine schooled in our ecosystem could later make itself useful in the real world. We consider a scenario in which the machine works as an assistant to Alice, an elderly person living alone. Bob is Alice's son, and he also interacts with the machine.

We assume that, as part of its training, the machine has been taught how to issue Internet commands and process their outcomes. In the example dialogue, we give a general idea of how the machine would interface to the Internet, without attempting to precisely define the syntax of this interaction. Most importantly, the Internet queries in the example are meant to illustrate how the machine does not need to store all the knowledge it needs to accomplish its duties, as it can retrieve useful information from the Web on demand, and reason about it.

Input:	*Output:*
Bob: I just spoke to the doctor, who said my mother needs to move for at least one hour per day, please make sure he does get enough exercise.	
...	
following conversation takes place the next day:	
	@Alice: Are you in the mood for some light physical exercise today?
Alice: Yes, but no more than 15 minutes, please.	
	@INTERNET: [query search engine for keywords *elderly, light activity, 15 minutes*]
shortly afterwards...	
	@Alice: I have downloaded a YouTube video with a 15-minute yoga routine, please watch it whenever you can.

Input:	*Output:*
a few hours later. . .	
Alice: I'm going out to buy groceries.	
	@INTERNET: [query search engine with keywords *average walking speed, elderly person*]
	@INTERNET: [search maps app for distance to grocery stores in Alice's neighborhood]
	@Alice: Hey, why don't you walk to the Yummy Food Market today? It should take you about 45 minutes to and from, so you'll get the rest of your daily exercise.
@Alice: Thanks for the suggestion.	

The communication-based intelligent machine should adapt to a whole range of tasks it was not explicitly programmed for. If needed, the user can give it further positive and negative reward to motivate it to change its behavior. This may be needed only rarely, as the machine should be shipped to the end users after it already mastered good language and communication abilities, and further development should mostly occur through language. For example, when the user says *No, don't do this again*, the machine will understand that repeating the same type of behavior might lead to negative reward, and it will change its course of action even when no explicit reward signal is given.

The range of tasks for intelligent machines can be very diverse: besides the everyday-life assistant we just considered, it could explain students how to accomplish homework assignments, gather statistical information from the Internet to help medical researchers, find bugs in computer programs, or even write programs on its own. Intelligent machines should extend our intellectual abilities in the same way the current computers already function as an extension to our memory. This should enable us to perform intellectual tasks beyond what is possible today.

4 Towards the development of intelligent machines

In this section, we will describe some of our ideas and opinions about how to build intelligent machines that would benefit from the learning environment we described. While we do not have a concrete proposal yet about how exactly such machines should be implemented, we will discuss some of the properties and components we think are needed to support the desired functionalities. We have no pretense of completeness, we simply want to provide some food for thought. As in the previous sections, we try to keep the complexity of the machine at the minimum, and only consider the properties that seem essential.

4.1 Types of learning

There are many types of behavior that we collectively call learning, and it is useful to distinguish between them to clarify further discussion. Suppose our goal is to build an intelligent machine working as a translator between two languages. First, we will teach the machine basic communication skills in our simulated environment so that it can react to requests given by the user. Then, we will start teaching it, by example, how various words are translated.

There are different kinds of learning happening here. To master basic communication skills, the machine will have to understand the concept of positive and negative reward, and develop complex strategies to deal with novel linguistic inputs. This may require discovery of algorithms, and the ability to remember facts, skills and even learning strategies.

Next, in order to translate, the machine needs to store pairs of words. The number of pairs is unknown and a flexible growing mechanism may be required. However, once the machine understands how to populate the dictionary with examples, the learning left to do is of a very simple nature: the machine does not have to update its learning strategy, but only to store and organize the incoming information into long-term memory using previously acquired skills. Finally, once the vocabulary memorization process is finished and the machine starts working as a translator, no further learning might be required, and the functionality of the machine can be fixed.

The more specialized and narrow the functionality of the machine is, the

less learning is required. This follows our intuition that, for very specialized forms of behavior, it is possible to program the solution manually. However, as we move from roles such as a simple translator of words, a calculator, a chess player, etc., to machines with open-ended goals, we need to rely more on general learning from a limited number of examples.

One can see the current state of the art in machine learning as being somewhere in the middle of this hierarchy. Tasks such as automatic speech recognition, classification of objects in images or machine translation are already too hard to be solved purely through manual programming, and the best systems rely on some form of statistical learning, where parameters of hand-coded models are estimated from large datasets of examples. However, the capabilities of state-of-the-art machine learning systems are severely limited, and only allow a small degree of adaptability of the machine's functionality. For example, a speech recognition system will never be able to perform speech translation by simply being instructed to do so – a human programmer is required to implement additional modules manually.

4.2 Long-term memory and compositional learning skills

We see a special kind of long-term memory as the key component of the intelligent machine. This long-term memory should be able to store facts and algorithms corresponding to learned skills, and be accessed on-demand. In fact, even the ability to learn should be seen as a set of skills that are stored in the memory. When the learning skills are triggered by the current situation, they should compose new persistent structures in the memory from the existing ones. Thus, the machine should have the capacity to extend itself.

Without being able to store previously learned facts and skills, the machine could not deal with rather trivial assignments, such as recalling the solution to a task that has been encountered before. Moreover, it is often the case that the solution to a new task is related to that of earlier ones. Consider for example the following sequence of tasks in our simulated environment:

- find and pick an apple;
- bring the apple back home;
- find two apples;

- find one apple and two bananas and bring them home.

Skills required to solve these tasks include:

- the ability to search around the current location;
- the ability to pick things;
- the ability to remember the location of home and return to it;
- the ability to understand what *one* and *two* means;
- the ability to combine the previous skills (and more) to deal with different requests.

The first four abilities correspond to simple facts or skills to be stored in memory: a sequence of symbols denoting something, the steps needed to perform a certain action, etc. The last ability is an example of a compositional *learning skill*, with the capability of producing new structures by composing together known facts and skills. Thanks to such learning skills, the machine will be able to combine several existing abilities to create a new one, often on the fly. In this way, a well-functioning intelligent machine will not need a myriad of training examples whenever it faces a slightly new request, but it could succeed given a single example of the new functionality. For example, when Teacher asks Learner to find one apple and two bananas and bring them home, if the Learner already understands all the individual abilities involved, it can retrieve the relevant compositional learning skill to put together a plan and execute it step by step. The Teacher may even call the new skill generated in this was *prepare breakfast*, and refer to it later as such. This should not require any further training of Learner, that would simply store the new skill together with its label in long-term memory.

4.3 Computational properties of intelligent machines

Another property of the intelligent machine that deserves discussion is the computational model that the machine will be based on. We are convinced that such model should be unrestricted, that is, able to represent any pattern in the data. Humans can think of and talk about algorithms without obvious limitations (although, to apply them, they might need to rely on forms of

external support, such as pencil and paper). A useful intelligent machine should be able to handle such algorithms as well.

A more precise formulation in the context of the theory of computation is that the intelligent machine needs to be based on a Turing-complete computational model. That is, it has to be able to represent any algorithm in fixed length, just like the Turing machine (the very fact that humans can describe Turing-complete algorithms shows that they are, in practical terms, Turing-complete). Note that there are many Turing-complete computational systems, and Turing machines in particular are a lot less efficient than some alternatives, e.g., Random Access Machines. Thus, we are not interested in building the intelligent machine around the concept of the Turing machine; we just aim to use a computational model that does not have obvious limitations in representing patterns.

A system that is weaker than Turing-complete cannot represent certain patterns in data efficiently, which in turn means it cannot truly learn them in a general sense. However, it is possible to memorize such complex patterns up to some finite level of complexity. Thus, even a computationally restricted system may appear to work as intended up to some level of accuracy, given that a sufficient number of training examples is provided.

For example, we may consider a sequence repetition problem. The machine is supposed to remember a sequence of symbols and reproduce it later. Further, let's assume the machine is based on a model with the representational power of finite state machines. Such system is not capable to represent the concept of storing and reproducing a sequence. However, it may appear to do so if we design our experiment imperfectly. Assume there is a significant overlap between what the machine sees as training data, and the test data we use to evaluate performance of the machine. A trivial machine that can function as a look-up table may appear to work, simply by storing and recalling the training examples. With an infinite number of the training examples, a look-up table based machine can appear to learn any regularity. It will work indistinguishably from a machine that can truly represent the concept of repetition; however, it will need to have an infinite size. Clearly, such memorization-based system will not perform well in our settings, as we aim to test the Learner's ability to generalize from a few examples.

Since there are many Turing-complete computational systems, one may wonder which one should be preferred as the basis for machine intelligence. We cannot answer this question yet, however we hypothesize that the most natural choice would be a system that performs computation in a parallel way,

using elementary units that can grow in number based on the task at hand. The growing property is necessary to support the long-term memory, if we assume that the basic units themselves are finite. An example of an existing computational system with many of the desired properties is the cellular automaton of Von Neumann et al. (1966). We might also be inspired by string rewriting systems, for example some versions of the L-systems (Prusinkiewicz and Lindenmayer, 2012).

An apparent alternative to our requirements would be to use a non-growing model with immensely large capacity. There is however an important difference: In a growing model, the new cells can be connected to those that spawned them, so that the model is naturally able to develop a meaningful topological structure based on functional connectivity. We conjecture that such structure would in itself contribute to learning in a crucial way. On the other hand, it is not clear if such topological structure can arise in the large-capacity unstructured model. Interestingly, some of the more effective machine-learning models available today, such as recurrent and convolutional neural networks, are characterized by (manually constrained) network topologies that are well-suited to the domains they are applied to.

5 Related ideas

Like any serious roadmap towards AI, ours owes a large debt to the seminal ideas of Turing (1950). Note that, while Turing's paper is most often cited for the "imitation game" proposal, there are other very interesting ideas in it, worthy of more attention from curious readers, especially in the last section on learning machines. Turing thought that a good way to construct an engine capable of passing his famous test would be to develop a *child machine*, and teach it further skills through various communication channels. These would include sparse rewards shaping the behavior of the child machine, and other information-rich channels such as language input from a teacher and sensory information.

We share Turing's goal of developing a child machine capable of independent communication through natural language, and we also stress the importance of sparse rewards. The main distinction between his and our vision is that Turing assumed that the child machine would be largely programmed (he gives an estimate of sixty programmers working on it for fifty years). We rather think of starting with a machine only endowed with very

elementary skills, and focus on the capability to learn as the crucial ability that needs to be developed. This further assumes educating the machine at first in a simulated environment where an artificial teacher will train it, as we outlined in our roadmap. We also diverge with respect to the imitation game, since the purpose of our intelligent machine is not to fool human judges into believing it is actually a real person. Instead, we aim to develop a machine that can perform a similar set of tasks that a human can do by using a computer, an Internet connection and the ability to communicate.

There has been a recent revival of interest in tasks measuring computational intelligence, spurred by the empirical advances of powerful machine-learning architectures such as multi-layered neural networks (LeCun et al., 2015), and by the patent inadequacy of the classic version of Turing test (Wikipedia, 2015b). For example, Levesque et al. (2012) propose to test systems on their ability to resolve coreferential ambiguities (*The trophy would not fit in the brown suitcase because it was too big... What was too big?*). Geman et al. (2015) propose a "visual" Turing test in which a computational system is asked to answer a set of increasingly specific questions about objects, attributes and relations in a picture (*Is there a person in the blue region? Is the person carrying something? Is the person interacting with any other object?*). Similar initiatives differ from ours in that they focus on a specific set of skills (coreference, image parsing) rather than testing if an agent can learn new skills. Moreover, these are traditional evaluation benchmarks, unlike the hybrid learning/evaluation ecosystem we are proposing.

The idea of developing an AI living in a controlled synthetic environment and interacting with other agents through natural language is quite old. The Blocks World of Winograd (1971) is probably the most important example of early research in this vein. The approach was later abandoned, when it became clear that the agents developed within this framework did not scale up to real-world challenges (see, e.g., Morelli et al., 1992). The knowledge encoded in the systems tested by these early simulations was manually programmed by their creators, since they had very limited learning capabilities. Consequently, scaling up to the real world implied manual coding of all the knowledge necessary to cope with it, and this proved infeasible. Our simulation is instead aiming at systems that encode very little prior knowledge and have strong capabilities to learn from data. Importantly, our plan is not to try to manually program all possible scripts our system might encounter later, as in some of the classic AI systems. We plan to script only the initial environment, in order to kickstart the machine's ability to learn

and adapt to different problems and scenarios. After the simulated environment is mastered, scaling up the functionality of our Learner will not require further manual work on scripting new situations, but will rather focus on integrating real world inputs, such as those coming from human users. The toy world itself is already designed to feature novel tasks of increasing complexity, explicitly testing the abilities of systems to autonomously scale up.

Still, we should not underestimate the dangers of synthetic simulations. Advanced tasks in our environment should directly address some weak points of early AI, such as dealing with very large object ontologies and vocabularies, noise robustness, massive ambiguity and the need for fuzzy, analogical reasoning. However, *simulating* the real world can only bring us so far, and we might end up overestimating the importance of some arbitrary phenomena at the expense of others, that might turn out to be more common in natural settings. A better strategy is to eventually bring reality into the picture. The aim of our toy world is to let an intelligent machine develop to the point to which it is able to cooperate with and learn from actual humans. Interaction with real-life humans should naturally lead the machine to deal with real-world problems. The issue of when exactly a machine trained in a controlled synthetic environment is ready to go out in the human world is open, and it should be explored empirically.

Our intelligent machine shares some of its desired functionalities with the current generation of automated personal assistants such as Apple's Siri ad Microsoft's Cortana. However, these are heavily engineered systems that aim to provide a natural language interface for human users to perform a varied but fixed set of tasks (similar considerations also apply to artificial human companions and digital pets such as Tamagotchi, see Wikipedia, 2015a). Such systems can be developed by defining the most frequent use cases, choosing those that can be solved with the current technology (e.g., book an air ticket, look at the weather forecast and set the alarm clock for tomorrow's morning), and implementing specific solutions for each such use case. Our intelligent machine is not intended to handle just a fixed set of tasks. As exemplified by the example in Section 3.3, the machine should be capable to learn efficiently how to perform tasks such as those currently handled by personal assistants, and more, just from interaction with the human user (without a programmer or machine learning expert in the loop).

Architectures for software agents, and more specifically *intelligent* agents, are widely studied in AI and related fields (Nwana, 1996; Russell and Norvig,

2009). We cannot review this ample literature here, in order to position our proposal precisely with respect to it. We simply remark that we are not aware of other architectures that are as centered on learning and communication as ours. Interaction also plays a central role in the study of multiagent systems (Shoham and Leyton-Brown, 2009). However, the emphasis in this research tradition is on how conflict resolution and distributed problem solving evolve in typically large groups of simple, mostly scripted agents. For example, traffic modeling is a classic application scenario for multiagent systems. This is very different from our emphasis on linguistic interaction for the purposes of training a single agent that should become independently capable of very complex behaviours.

Tenenbaum (2015), like us, emphasizes the need to focus on basic abilities that form the core of intelligence. However, he takes naive physics problems as the starting point, and discusses specific classes of probabilistic models, rather than proposing a general learning scenario. There are also some similarities between our proposal and the research program of Luc Steels (e.g., Steels, 2003, 2005), who lets robots evolve vocabularies and grammatical constructions through interaction in a situated environment. However, on the one hand his agents are actual robots subject to the practical hardware limitations imposed by the need to navigate a complex natural environment from the start; on the other, the focus of the simulations is narrowly on language acquisition, with no further aim to develop broadly intelligent agents.

We have several points of contact with the semantic parsing literature, such as navigation tasks in an artificial world (MacMahon et al., 2006) and reward-based learning from natural language instructions (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013). Some goals pursued in this area, such as developing agents that learn to execute instructions in natural environments by interacting with humans (Thomason et al., 2015) or improving performance on real-life video-games by consulting the instruction manual (Branavan et al., 2012), are certainly landmarks of intelligence. However, current systems achieve these impressive feats by exploiting architectures tuned to the specific tasks at hand, and they rely on a fair amount of hard-wired expert knowledge, in particular about language structures (although recent work is moving towards a more knowledge-lean direction, see for example Narasimhan et al., 2015, who train a neural network to play text-based adventure games using only text descriptions as input and game reward as signal). Our framework is meant to encourage the development of systems that should eventually be able to perform similar tasks, but getting

there incrementally by learning from their environment a set of simpler skills, and how to creatively merge them to tackle more ambitious goals.

The last twenty years have witnessed several related proposals on learning to learn (Thrun and Pratt, 1997), lifelong learning (Silver et al., 2013) and continual learning (Ring, 1997). Much of this work is theoretical in nature and focuses on algorithms rather than on empirical challenges for the proposed models. Still, the general ideas being pursued are in line with our program. Ring (1997), in particular, defines a continual-learning agent whose experiences "occur sequentially, and what it learns at one time step while solving one task, it can use later, perhaps to solve a completely different task." Ring's desiderata for the continual learner are remarkably in line with ours. It is "an autonomous agent. It senses, takes actions, and responds to the rewards in its environment. It learns behaviors and skills while solving its tasks. It learns incrementally. There is no fixed training set; learning occurs at every time step; and the skills the agent learns now can be used later. It learns hierarchically. Skills it learns now can be built upon and modified later. It is a black box. The internals of the agent need not be understood or manipulated. All of the agent's behaviors are developed through training, not through direct manipulation. Its only interface to the world is through its senses, actions, and rewards. It has no ultimate, final task. What the agent learns now may or may not be useful later, depending on what tasks come next." Ring's example, a set of increasingly complex mazes where an agent must discover food by reinforcement learning, is also somewhat related to our setup.

Mitchell et al. (2015) discuss NELL, the most fully realized concrete implementation of a lifelong learning architecture. NELL is an agent that has been "reading the Web" for several years to extract a large knowledge base. Emphasis is on the never-ending nature of the involved tasks, on their incremental refinement based on what NELL has learned, and on sharing information across tasks. In this latter respect, this project is close to multi-task learning (Ando and Zhang, 2005; Caruana, 1997; Collobert et al., 2011), that focuses on the idea of parameter sharing across tasks. It is likely that a successful learner in our framework will exploit similar strategies, but our current focus lies on defining the tasks, rather than on how to pursue them.

Bengio et al. (2009) propose the related idea of curriculum learning, whereby training data for a single task are ordered according to a difficulty criterion, in the hope that this will lead to better learning. This is motivated by the observation that humans learn incrementally when developing

complex skills, an idea that has also previously been studied in the context of recurrent neural network training by Elman (1990). Note that we expect the intelligent machine to develop incrementally more complex *skills* during its lifetime; this does not necessarily require the training *data* to be precisely ordered, which was the focus of these previous works. The idea of incremental learning, motivated by the same considerations as in the papers we just mentioned, also appears in Solomonoff (2002), a work which has much earlier roots in research on program induction (Solomonoff, 1964, 1997; Schmidhuber, 2004). Genetic programming (Poli et al., 2008) also focuses on the reuse of previously found sub-solutions, speeding up the search procedure in this way. Our proposal is also related to that of Bottou (2014), in its vision of compositional machine learning.

We share many ideas with the reinforcement learning framework (Sutton and Barto, 1998). In reinforcement learning, the agent chooses actions in an environment in order to maximize some cumulative reward over time. Reinforcement learning is particularly popular for problems where the agent can collect information only by interacting with the environment. Given how broad this definition is, our framework could be considered as a particular instance of it. Our proposal is however markedly different from standard reinforcement learning work(Kaelbling et al., 1996) in several respects. For example, we focus on strategies to encourage agents to solve tasks by reusing previously learned knowledge, we aim at limiting the number of trials an agent has to accomplish a certain goal, we maximize average reward, which favors efficient agents, and we allow the agent to interact with the environment between tasks to prepare itself for the next task.

Mnih et al. (2015) recently presented a single neural network architecture capable of learning a set of classic Atari games using only pixels and game scores as input (see also the related idea of "general game playing", e.g., Genesereth et al., 2005). We pursue a similar goal of learning from a low-level input stream and reward. However, unlike these authors, we do not aim for a single architecture that can, disjointly, learn an array of separate tasks, but for one that can incrementally build on skills learned on previous tasks to perform more complex ones. Moreover, together with reward, we emphasize linguistic interaction as a fundamental mean to foster skill extension.

Mikolov (2013) originally discussed a preliminary version of the incremental task-based approach we are more fully outlying here. In a similar spirit, Weston et al. (2015) present a set of question answering tasks based on synthetically generated stories. They also want to foster non-incremental

progress in AI, but their approach differs from ours in several respects. Again, there is no notion of interactive, language-mediated learning, a classic train/test split is enforced, and the tasks are not designed to encourage compositional skill learning. Moreover, while Weston and colleagues emphasize that the same system should be used in all tasks, there is no clear overarching purpose in light of which the proposed tasks would form a coherent whole. Although we also envisage separate tasks to be pursued within the ecosystem, our end purpose to develop an agent able to enter in collaborative endeavors with humans naturally imposes an overarching goal to task design, that is, we want to develop clusters of interrelated tasks that support high-level communication.

One could think of solving sequence-manipulation problems such as those presented in this paper with relatively small extensions of established machine learning techniques (Graves et al., 2014; Grefenstette et al., 2015; Joulin and Mikolov, 2015). While such techniques might appear to work, this conclusion may be overly optimistic, as we discussed in the previous section. For simple tasks that involve only a small, finite number of configurations, one could be apparently successful even just by using a look-up table storing all possible combinations of inputs and outputs. The above mentioned works that aim to learn algorithms from data then use a long-term memory (e.g., some stacks) only to store the data, but not the learned algorithms. Thus, such approaches fail to work in environments where solutions to new tasks are composed of already learned algorithms.

The same criticism holds for models that try to learn certain algorithms by using an architecture with a strong prior towards their discovery, but not general enough to represent even small modifications. To give an example from our own work: a recurrent neural network augmented with a stack structure can form a simple kind of long-term memory and learn to memorize and repeat sequences in the reversed order, but not in the original one (Joulin and Mikolov, 2015). We expect a valid solution to the algorithmic learning challenge to utilize a small number of training examples, and to learn tasks that are closely related at an increasing speed, i.e., to require less and less examples to master new skills that are related to what is already known. We are not aware of any current technique addressing these issues, which were the very reason why algorithmic tasks were originally proposed by Mikolov (2013). We hope that this paper will motivate the design of the genuinely novel methods we need in order to develop intelligent machines.

6 Conclusion

We defined basic desiderata for an intelligent machine, stressing learning and communication as its fundamental abilities. Contrary to common practice in current machine learning, where the focus is on modeling single skills in isolation, we believe that all aspects of intelligence should be holistically addressed within a single system.

We proposed a simulated environment that requires the intelligent machine to acquire new facts and skills through communication in natural language. In this environment, the machine must learn to perform increasingly more ambitious tasks, being naturally induced to develop complex linguistic and reasoning abilities.

We also presented some conjectures on the properties of the computational system that the intelligent machine may be based on. This includes learning of algorithmic patterns from a few examples without strong supervision, and development of long term memory that would store both data and learned skills. We tried to put this in contrast with currently accepted paradigms in machine learning, to show that current methods are far from adequate, and we must strive to develop non-incrementally novel techniques.

Acknowledgments

We thank Gemma Boleda, Léon Bottou, Yann LeCun, Gabriel Synnaeve, Arthur Szlam, Nicolas Usunier, Laurens van der Maaten, Wojciech Zaremba and others from the Facebook AI Research team for many stimulating discussions. An early version of this proposal has been discussed in several research groups since 2013 under the name *Incremental learning of algorithms* (Mikolov, 2013).

References

Ando, R. and Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 5:1817–1853.

Artzi, Y. and Zettlemoyer, L. (2013). Weakly supervised learning of semantic

parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of ICML*, pages 41–48, Montreal, Canada.

Bottou, L. (2014). From machine learning to machine reasoning: an essay. *Machine Learning*, 94:133–149.

Branavan, S., Silver, D., and Barzilay, R. (2012). Learning to win by reading manuals in a Monte-Carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704.

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28:41–75.

Chen, D. and Mooney, R. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of AAAI*, pages 859–865, San Francisco, CA.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Fodor, J. (1975). *The Language of Thought*. Crowell Press, New York.

Geman, D., Geman, S., Hallonquist, N., and Younes, L. (2015). Visual Turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 112(12):3618–3623.

Genesereth, M., Love, N., and Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. `http://arxiv.org/abs/1410.5401`.

Grefenstette, E., Hermann, K., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Proceedings of NIPS*, Montreal, Canada. In press.

Haugeland, J. (1985). *Artificial Intelligence: The Very Idea*. MIT Press, Cambridge, MA.

Hofstadter, D. and Sander, E. (2013). *Surfaces and Essences: Analogy as the Fuel and Fire of Thinking*. Basic Books, New York.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Proceedings of NIPS*, Montreal, Canada. In press.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285.

Lakoff, G. and Johnson, M. (1999). *Philosophy in the Flesh: The Embodied Mind and Its Challenge to Western Thought*. Basic Books, New York.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–444.

Levesque, H. J., Davis, E., and Morgenstern, L. (2012). The Winograd schema challenge. In *Proceedings of KR*, pages 362–372, Rome, Italy.

Louwerse, M. (2011). Symbol interdependency in symbolic and embodied cognition. *Topics in Cognitive Science*, 3:273–302.

MacMahon, M., Stankiewicz, B., and Kuipers, B. (2006). Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of AAAI*, pages 1475–1482, Boston, MA.

Mikolov, T. (2013). Incremental learning of algorithms. Unpublished manuscript.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Mishra, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (2015). Never-ending learning. In *Proceedings of AAAI*, pages 2302–2310, Austin, TX.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Morelli, R., Brown, M., Anselmi, D., Haberlandt, K., and Lloyd, D., editors (1992). *Minds, Brains, and Computers: Perspectives in Cognitive Science and Artificial Intelligence*. Ablex, Norwood, NJ.

Narasimhan, K., Kulkarni, T., and Barzilay, R. (2015). Language understanding for text-based games using deep reinforcement learning. In *Proceedings of EMNLP*, pages 1–11, Lisbon, Portugal.

Nwana, H. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(2):1–40.

Poli, R., Langdon, W., McPhee, N., and Koza, J. (2008). A field guide to genetic programming. `http://www.gp-field-guide.org.uk`.

Prusinkiewicz, P. and Lindenmayer, A. (2012). *The algorithmic beauty of plants*. Springer Science & Business Media.

Ring, M. (1997). CHILD: A first step towards continual learning. *Machine Learning*, 28:77–104.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach, 3d ed.* Pearson Education, New York.

Schmidhuber, J. (2004). Optimal ordered problem solver. *Machine Learning*, 54(3):211–254.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems*. Cambridge University Press, Cambridge.

Silver, D., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *Proceedings of the AAAI Spring Symposium on Lifelong Machine Learning*, pages 49–55, Stanford, CA.

Solomonoff, R. J. (1964). A formal theory of inductive inference. Part I. *Information and control*, 7(1):1–22.

Solomonoff, R. J. (1997). The discovery of algorithmic probability. *Journal of Computer and System Sciences*, 55(1):73–88.

Solomonoff, R. J. (2002). Progress in incremental machine learning. In *NIPS Workshop on Universal Learning Algorithms and Optimal Search, Whistler, BC*. Citeseer.

Steels, L. (2003). Social language learning. In Tokoro, M. and Steels, L., editors, *The Future of Learning*, pages 133–162. IOS, Amsterdam.

Steels, L. (2005). What triggers the emergence of grammar? In *Proceedings of EELC*, pages 143–150, Hatfield, UK.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Tenenbaum, J. (2015). Cognitive foundations for knowledge representation in AI. Presented at the AAAI Spring Symposium on Knowledge Representation and Reasoning.

Thomason, J., Zhang, S., Mooney, R., and Stone, P. (2015). Learning to interpret natural language commands through human-robot dialog. In *Proceedings IJCAI*, pages 1923–1929, Buenos Aires, Argentina.

Thrun, S. and Pratt, L., editors (1997). *Learning to Learn*. Kluwer, Dordrecht.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59:433–460.

Von Neumann, J., Burks, A. W., et al. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015). Towards AI-complete question answering: A set of prerequisite toy tasks. `http://arxiv.org/abs/1502.05698`.

Whiten, A., editor (1991). *Natural Theories of Mind*. Blackwell, Malden, MA.

Wikipedia (2015a). Artificial human companion. `https://en.wikipedia.org/w/index.php?title=Artificial_human_companion&oldid=685507143`. Accessed 15-October-2015.

Wikipedia (2015b). Turing test. https://en.wikipedia.org/w/index.php?title=Turing_test&oldid=673582926. Accessed 30-July-2015.

Winograd, T. (1971). Procedures as a representation for data in a computer program for understanding natural language. Technical Report AI 235, Massachusetts Institute of Technology.