1 The Bitter Lesson

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers belated learning this bitter lesson, and it is instructive to review some of the most prominent.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that "brute force" search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to win and were disappointed when they did not.

A similar pattern of research progress was seen in computer Go, only delayed by a further 20 years. Enormous initial efforts went into avoiding search by taking advantage of human knowledge, or of the special features of the game, but all those efforts proved irrelevant, or worse, once search was applied effectively at scale. Also important was the use of learning by self play to learn a value function (as it was in many other games and even in chess, although learning did not play a big role in the 1997 program that first beat a world champion). Learning by self play, and learning in general, is like search in that it enables massive computation to be brought to bear. Search and learning are the two most important classes of techniques for utilizing massive amounts of computation in AI research. In computer Go, as in computer chess, researchers' initial effort was directed towards utilizing human understanding (so that less search was needed) and only much later was much greater success had by embracing search and learning.

In speech recognition, there was an early competition, sponsored by DARPA, in 1970. Entrants included a host of special methods that took advantage of human knowledge—knowledge of words, of phonemes, of the human vocal tract, etc. On the other side were newer methods that were more statistical in nature and did much

more computation, based on hidden Markov models (HMMs). Again, the statistical methods won out over the human-knowledge-based methods. This led to a major change in all of natural language processing, gradually over decades, where statistics and computation came to dominate the field. The recent rise of deep learning in speech recognition is the most recent step in this consistent direction. Deep learning methods rely even less on human knowledge, and use even more computation, together with learning on huge training sets, to produce dramatically better speech recognition systems. As in the games, researchers always tried to make systems that worked the way the researchers thought their own minds worked—they tried to put that knowledge in their systems—but it proved ultimately counterproductive, and a colossal waste of researcher's time, when, through Moore's law, massive computation became available and a means was found to put it to good use.

In computer vision, there has been a similar pattern. Early methods conceived of vision as searching for edges, or generalized cylinders, or in terms of SIFT features. But today all this is discarded. Modern deep-learning neural networks use only the notions of convolution and certain kinds of invariances, and perform much better.

This is a big lesson. As a field, we still have not thoroughly learned it, as we are continuing to make the same kind of mistakes. To see this, and to effectively resist it, we have to understand the appeal of these mistakes. We have to learn the bitter lesson that building in how we think we think does not work in the long run. The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning. The eventual success is tinged with bitterness, and often incompletely digested, because it is success over a favored, human-centric approach.

One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*. The second general point to be learned from the bitter lesson is that the actual contents of minds are tremendously, irredeemably complex; we should stop trying to find simple ways to think about the contents of minds, such as simple ways to think about space, objects, multiple agents, or symmetries. All these are part of the arbitrary, intrinsically-complex, outside world. They are not what should be built in, as their complexity is endless; instead we should build in only the meta-methods that can find and capture this arbitrary complexity. Essential to these methods is that they can find good approximations, but the search for them should be by our methods, not by us. We want AI agents that can discover like we can, not which contain what we have discovered. Building in our discoveries only makes it harder to see how the discovering process can be done.

2 Principles

The bitter lesson should teach us to treat the problem of AI in a very general way, leveraging search and learning. We should not focus on human knowledge about special cases. We should use the data and the regularities in the data. Thus we in the Core RL project identified and committed to a list of principles, or desiderata, that substantially constrain and guide our research. We seek methods that are:

General Domain knowledge is *not* of interest in Core RL. It may be used, pragmatically, but it is not a subject of study. We do not want to spend time thinking about what domain knowledge we should build in, about how it should be provided, about curricula or demonstrations for providing it, or about how the rewards should be set to achieve a human goal. All these things are of obvious importance to any deployed AI, but not to core RL. They will influence performance, but are not part of core intelligence nor bottlenecks to achieving the core abilities of intelligence.

Online and incremental Learning should proceed online from a single stream of experience.

Fast responding The agent's action selections should depend on its input with very low latency. A premium is placed on rapid action, implying that no extensive computation can stand between observation and action. All extensive computations can affect only later actions.

Of low learning complexity, that scale well Memory, per-time-step compute, and communication costs should be

- order the number of weights or modifiable parameters (e.g., not order the square of the number of parameters).
- non-increasing with experience. This is sometimes called *strong incrementality*.
- non-increasing with the *temporal span* of predictions and memories. The temporal span of a prediction is the maximum number of time step elapsing between when a prediction is made and when the actual outcome is known. For example, a value function predicts the cummulative reward over a potentially infinite future—infinite span—yet can be learned by temporal-difference or eligibility trace methods of modest complexity. Predicting ahead by exactly k time steps, on the other hand, requires computational resources (including memory) that scales with k. The temporal span of a memory is the maximum number of time steps elapsing between when an event happens and when it is recalled. If a memory consist of the last k observations, then its span is k and its complexity scales with k. If backpropagation-through-time is done over a length k window, then its memory and possibly its computation scales with k.

Temporally symmetric All times should be the same; there should be no special times. This rules out, for example, methods like optimistic initialization. It also rules out the way normal backpropagation relies on initial small random weights in the hidden units. To the extent that these are necessary to get good performance, backpropagation is not temporally symmetric.

Much smaller than the environment (aperture principle) The total complexity of the agent is much less than the total complexity of the environment (which, after all, may contains many other agents).

These constraints limit what we can do, which might be seen as bad, but they also focus our efforts. They rule out things that are interesting but not essential and that are likely to be distractions from core abilities of AI agents. They also set constraints on computational resources that may seem too limiting. There are certainly interesting problem domains where these limitations are not essential. However, the limitations do not preclude intelligence, as they apply to natural intelligence. Such strong constraints that do not rule out the primary use cases can be very useful at this stage of the search for AI agents.

3 Issues

What is an issue? It is a deliberately weak word, but not without meaning. The ideal issue is a scientific hypothesis or question. But it might be less than that. There might be some odd thing that you have noticed while doing experiments, or in some corner in the theory of reinforcement learning. The issue, then, is whether or not that thing is real. Does it represent a leverage point that leads to improved algorithms or theory, or is it indeed just a quirk that is best ignored? Such things are issues. Issues are loose ends in the space of research questions.

Issues can be large or small. They might need a lot of explanation or they may be compact and crisp to state. The ideal case would be an issue that could be addressed, assessed, and resolved with a small experiment. But the most important issues often cannot yet be handled entirely that way. In fact, in many cases an issue is rather vague and the process of making it less vague gives rise to multiple other issues (some of which may be smaller and crisper). In the Core RL project, we are particularly interested in issues that are going to impact the attempt to understand and create intelligence in the relatively long term (five years or more).

Below we list a number of issues that we see as ripe for Core RL research. It is too ambitious at this point to attempt an exhaustive list. There are an infinite number of issues; we will always be able to think of more, even if we resolve all that we can think of now. More to the point, we will easily be able to list more scientific issues than can be resolved in the Core RL project even if it lasts many years. What then is our goal here? It is to illustrate an exemplary methodology by identifying issues and pursuing a few of them. To this end we wish to identify a largish set of illustrative issues. These will make more plain the idea of an issue oriented methodology and may provide the seeds of subsequent efforts, large or small, to resolve them. Then we

identify a few issues to be taken further within the Core RL project to illustrate the rest of the steps of the methodology.

Here, then, is our initial list of issues:

Distributional Reinforcement Learning DRL has had some successes but it is not well understood when or why. This is an obvious area in which to apply the Core RL methodology. Some small experiments with simpler algorithms could test aspects of full DRL.

Getting online learning to work It is often thought that learning offline in batches is needed to get reinforcement learning to work well. But is this true? Is it just for nonlinear nets, or it more generally true? Can we make some simple illustrations of the problem? If so, then can we use these to try out, test, and invent new solution methods?

In particular, can we show this problem for policy gradient methods with linear function approximation?

Learning with less domain knowledge While we like to think of our algorithms as general solution principles, in practice, we often rely on significant amounts of domain knowledge in order to get them to work on challenging domains and benchmarks. Domain knowledge takes various forms (e.g., the use of fixed discount factors which differ from the true objective we are trying to optimize, reward shaping, and large numbers of hyper-parameters pretuned specifically for the domain), and it is often unclear how much our algorithms rely on this domain knowledge to achieve good behaviour on tasks of interest.

Can we reduce our reliance on domain heuristics, by replacing them with more general learned solutions? What's the impact on performance on the domains these heuristics were defined for? What's the impact on the generality of our solutions?

Learning about one thing interferes with learning about another Our typical agents use one (large) function approximator to serve multiple purposes, like predicting values and choosing actions, but sharing part of its capacity between them. In the future, we may want to scale the number of heads/losses/purposes even further (Horde, UVFAs, Unreal, Predictron, FuN, etc.). It is possible that the current default solution of a scalar trade-off between losses is not good enough, and that we need a better understanding of interference and the allocation of the network capacity, so that one purpose does not drown out learning on others, or even undo learning progress on others.

Time, timing, and temporal extension in learning For general learning, we have reason to believe that, to make good decisions, agents may need to take into account the time between events, the time until a future change in signals, or the time since different signals have been observed. Animal learning literature suggests this hypothesis is worth testing. As one example, "time cells," akin to

place cells, have been identified and shown to possibly form a detailed representation of an animal's passage through time. In practice, we often create new signals or system state information (e.g., decaying traces of signals, or trace conditioning) to help our computational agents use the flow of time as a foundation for making decisions and sculpting policies in search of learning progress or reward. Previous work in robotics and prosthetics (Pilarski et al. 2012) has relied heavily on trace conditioning to make General Value Function (GVF) predictions possible and reliable in real-world environments. This so far has required the agent's designer to choose different types or forms of traces based on their understanding of the environment the agent will act within. Such hand-crafted representations of time has proved functional, but we expect a more general, agent-driven approach will be more suitable for general learning agents. We are interested in determining if agents do indeed need to take into account the flow of time to be successful, and, if so, how the flow of time is best captured by an agent (e.g., as state, as a separate algorithmic process, or as a natural consequence of existing learning mechanisms).

Timescale selection for incrementally computing average reward Ideally the method of calculating average reward would not impact the stability of a learning algorithm in a continuing (as opposed to episodic) setting. However, in practice certain methods for incrementally calculating the average reward have been observed to be connected to unstable, slow, or otherwise undesirable learning performance. It is unclear if this undesirable behaviour is repeatable and reproducible, and, if so, what approach to average-reward computation will lead to learning behaviour that lives up to our theoretical understanding for average reward. We are exploring the relationship between different average-reward-calculation approaches and other factors such as eligibility traces, function approximation, and non-Markov environments.

Issues in Planning with a Learned Model There are at least three interrelated issues involved in planning with a learned model: 1) discovering and learning agent state, 2) discovering and learning a model of the dynamics (state to state), and 3) planning effectively and efficiently with the model. All three issues are strongly interlinked. All of them are needed in order to properly/fully assess each component: to assess state one needs to see what one can predict from it; to assess the state-to-state dynamics, one needs the state; and to assess planning one needs the model, and one needs a changing world. A separate report presents some of the details of a new approach to this cluster of interrelated issues (Sutton 2018).