4. Game-Playing Machines

Alongside the generative adversarial networks introduced in Chapter 3, deep *reinforcement* learning has produced some of the most surprising artificial-neural-network advances, including the lion's share of the headline-grabbing "artificial intelligence" breakthroughs of recent years. In this chapter, we introduce what reinforcement learning is as well as how its fusion with deep learning has enabled machines to meet or surpass human-level performance on a diverse range of complex challenges, including Atari video games, the board game Go, and subtle physical-manipulation tasks.

DEEP LEARNING, AI, AND OTHER BEASTS

Earlier in this book, we introduced deep learning with respect to vision (Chapter 1), language (Chapter 2), and the generation of novel "art" (Chapter 3). In doing this, we've loosely alluded to deep learning's relationship to the concept of artificial intelligence. At this stage, as we begin to cover deep reinforcement learning, it is worthwhile to define these terms more thoroughly as well as the terms' relationships to one another. As usual, we will be assisted by visual cues—in this case, the Venn diagram in Figure 4.1.

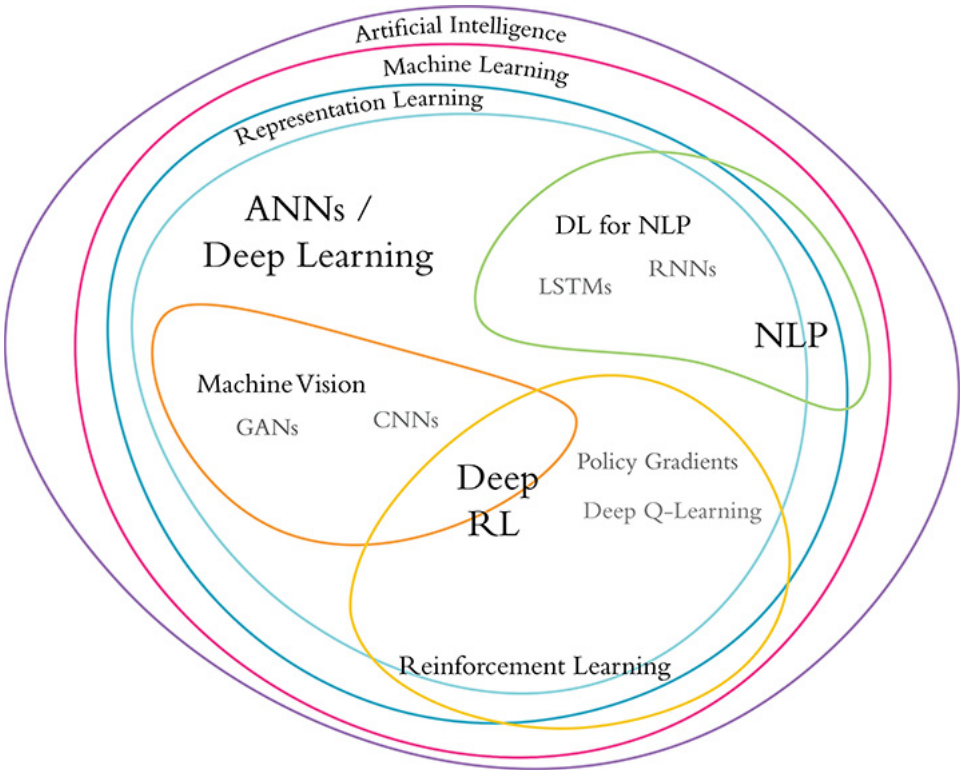


Figure 4.1 Venn diagram showing the relative positioning of the major concepts covered over the course of this book

Artificial Intelligence

Artificial intelligence is the buzziest, vaguest, and broadest of the terms we cover in this section. Taking a stab at a technical definition regardless, a decent one is that AI involves a machine processing information from its surrounding environment and then factoring that information into decisions toward achieving some desired outcome. Perhaps given this, some consider the goal of AI to be the achievement of "general intelligence"—intelligence as it is generally referred to with respect to broad reasoning and problem-solving capabilities. [1] In practice and particularly in the popular press, "AI" is used to describe any cutting-edge machine capability. Presently, these capabilities include voice recognition, describing what's happening in a video, question-answering, driving a car, industrial robots that mimic human exemplars in the factory, and dominating humans at "intuition-heavy" board games like Go. Once an AI capability becomes commonplace (e.g., recognizing handwritten digits, which was cutting-edge in the 1990s; see Chapter 1), the "AI" moniker is typically dropped by the popular press for that capability such that the goalposts on the definition of AI are always moving.

1 . Defining "intelligence" is not straightforward, and the great debate on it is beyond the scope of this book. A century-old definition of the term that we find amusing and that still today has some proponents among contemporary experts is that "intelligence is whatever IQ tests measure." See, for example, van der Mass, H., et al. (2014). Intelligence is what the intelligence test measures. Seriously. *Journal of Intelligence*, *2*, 12–15.

Machine Learning

Machine learning is a subset of AI alongside other facets of AI like robotics. Machine learning is a field of computer science concerned with setting up software in a manner so that the software can recognize patterns in data without the programmer needing to explicitly dictate how the software should carry out all aspects of this recognition. That said, the programmer would typically have some insight into or hypothesis about how the problem might be solved, and would thereby provide a rough model framework and relevant data such that the learning software is well prepared and well equipped to solve the problem. As depicted in Figure 1.12 and discussed time and again within the earlier chapters of this book, machine learning traditionally involves cleverly—albeit manually, and therefore laboriously—processing raw inputs to extract features that jibe well with data-modeling algorithms.

Representation Learning

Peeling back another layer of the Figure 4.1 onion, we find *representation learning*. This term was introduced at the start of Chapter 2, so we don't go into much detail again here. To recap briefly, representation learning is a branch of machine learning in which models are constructed in a way that—provided they are fed enough data—they learn features (or *representations*) automatically. These learned features may wind up being both more nuanced and more comprehensive than their manually curated cousins. The trade-off is that the learned features might not be as well understood nor as straightforward to explain, although academic and industrial researchers alike are increasingly tackling these hitches. [2]

2 . For example, see Kindermans, P.-J., et al. (2018). Learning how to explain neural networks: PatternNet and PatternAttribution. *International Conference on Learning Representations*.

Artificial Neural Networks

Artificial neural networks (ANNs) dominate the field of representation learning today. As was touched on in earlier chapters and will be laid bare in Chapter 6, artificial neurons are simple algorithms inspired by biological brain cells, especially in the sense that individual neurons—whether biological or artificial—receive input from many other neurons, perform some computation, and then produce a single output. An artificial neural network, then, is a collection of artificial neurons arranged so that they send and receive information between each other. Data (e.g., images of handwritten digits) are fed into an ANN, which processes these data in some way with the goal of producing some desired result (e.g., an accurate guess as to what digits are represented by the handwriting).

Deep Learning

Of all the terms in Figure 4.1, *deep learning* is the easiest to define because it's so precise. We have mentioned a couple of times already in this book that a network composed of at least a few layers of artificial neurons can be called a deep learning network. As exemplified by the classic architectures in Figures 1.11 and 1.17; diagramed simply in Figure 4.2; and fleshed out fully in Chapter 7, deep learning networks have a total of five or more layers with the following structure:

- A single *input* layer that is reserved for the data being fed into the network.

- Three or more *hidden* layers that learn representations from the input data. A general-purpose and frequently used type of hidden layer is the *dense* type, in which all of the neurons in a given layer can receive information from each of the neurons in the previous layer (it is apt, then, that a common synonym for "dense layer" is *fully connected layer*). In addition to this versatile hidden-layer type, there is a cornucopia of specialized types for particular use cases; we touch on the most popular ones as we make our way through this section.

- A single *output* layer that is reserved for the values (e.g., predictions) that the network yields.

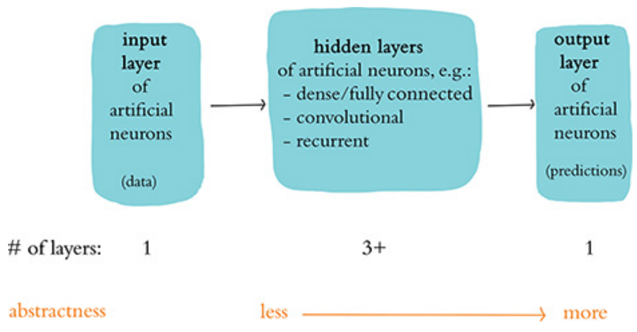

input layer of artificial neurons
(data)
hidden layers of artificial neurons, e.g.:
- dense/fully connected
- convolutional
- recurrent
output layer of artificial neurons
(predictions)
# of layers:
1
3+
1
abstractness
less
more

Figure 4.2 Generalization of deep learning model architectures

With each successive layer in the network being able to represent increasingly abstract, nonlinear recombinations of the previous layers, deep learning models with fewer than a dozen layers of artificial neurons are often sufficient for learning the representations that are of value for a given problem being solved with a given dataset. That said, deep learning networks with hundreds or even upwards of a thousand layers have in occasional circumstances been demonstrated to provide value. [3]

3 . For example, see He, K., et al. (2016). Identity mappings in deep residual networks. *arXiv:1603.05027*.

As rapidly improving accuracy benchmarks and countless competition wins since AlexNet's 2012 victory in the ILSVRC (Figure 1.15) have demonstrated, the deep learning approach to modeling excels at a broad range of machine learning tasks. Indeed, with deep learning driving so much of the contemporary progress in AI capabilities, the words "deep learning" and "artificial intelligence" are used essentially interchangeably by the popular press.

Let's move inside the deep learning ring of Figure 4.1 to explore classes of tasks that deep learning algorithms are leveraged for: machine vision, natural language processing, and reinforcement learning.

Machine Vision

Via analogy to the biological vision system, Chapter 1 introduced *machine vision*. There we focused on object recognition tasks such as distinguishing handwritten digits or breeds of dogs. Other prominent examples of applications that involve machine vision algorithms include self-driving cars, face-tagging suggestions, and phone unlocking via face recognition on smartphones. More broadly, machine vision is relevant to any AI that is going to need to recognize objects by their appearance at a distance or navigate a real-world environment.

Convolutional neural networks (ConvNets or CNNs for short) are a prominent type of deep learning architecture in contemporary machine vision applications. A CNN is any deep learning model architecture that features hidden layers of the *convolutional* type. We mentioned convolutional layers with respect to Ian Goodfellow's generative adversarial network results in Figure 3.2; we will detail and deploy them in Chapter 10.

Natural Language Processing

In Chapter 2, we covered language and natural language processing. Deep learning doesn't dominate natural language applications as comprehensively as it does machine vision applications, so our Venn diagram in Figure 4.1 shows NLP in both the deep learning region as well as the broader machine learning territory. As depicted by the timeline in Figure 2.3, however, deep learning approaches to NLP are beginning to overtake traditional machine learning approaches in the field with respect to both efficiency and accuracy. Indeed, in particular NLP areas like voice recognition (e.g., Amazon's Alexa or Google's Assistant), machine translation (including real-time voice translation over the phone), and aspects of Internet search engines (like predicting the characters or words that will be typed next by a user), deep learning already predominates. More generally, deep learning for NLP is relevant to any AI that interacts via natural language—be it spoken or typed—including to answer a complex series of questions automatically.

A type of hidden layer that is incorporated into many deep learning architectures in the NLP sphere is the *long short-term memory* (LSTM) cell, a member of the *recurrent neural network* (RNN) family. RNNs are applicable to any data that occur in a sequence such as financial time series data, inventory levels, traffic, and weather. We expound on RNNs, including LSTMs, in Chapter 11 when we incorporate them into predictive models involving natural language data. These language examples provide a firm foundation even if you're primarily seeking to apply deep learning techniques to the other classes of sequential data.

THREE CATEGORIES OF MACHINE LEARNING PROBLEMS

The one remaining section of the Venn diagram in Figure 4.1 involves reinforcement learning, which is the focus of the rest of this chapter. To introduce reinforcement learning, we contrast it with the two other principal categories of problems that machine learning algorithms are often leveraged to tackle: supervised and unsupervised learning problems.

Supervised Learning

In *supervised learning* problems, we have both an *x* variable and a *y* variable, where:

- *x* represents the data we're providing as input into our model.

- *y* represents an outcome we're building a model to predict. This *y* variable can also be called a *label*.

The goal with supervised learning is to have our model learn some function that uses *x* to approximate *y*. Supervised learning typically involves either of two types:

- *Regression*, where our *y* is a *continuous variable*. Examples include predicting the number of sales of a product, or predicting the future price of an asset like a home (an example we provide in Chapter 9) or a share in an exchange-listed company.

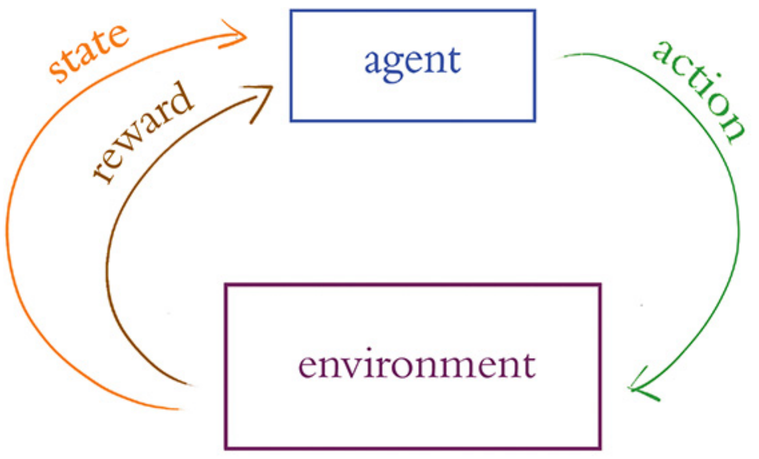
- *Classification*, where our *y*-values consist of labels that assign each instance of *x* to a particular category. In other words, *y* is a so-called *categorical variable*. Examples include identifying handwritten digits (you will code up models that do this in Chapter 10) or predicting whether someone who has reviewed a film loved it or loathed it (as you'll do in Chapter 11).

Unsupervised Learning

Unsupervised learning problems are distinguishable from supervised learning problems by the absence of a label *y*. Ergo, in unsupervised learning problems, we have some data *x* that we can put into a model, but we have no outcome *y* to predict. Rather, our goal with unsupervised learning is to have our model discover some hidden, underlying structure within our data. An often-used example is that of grouping news articles by their theme. Instead of providing a predefined list of categories that the news articles belong to (politics, sports, finance, etc.), we configure the model to group those with similar topics for us automatically. Other examples of unsupervised learning include creating a word-vector space (see Chapter 2) from natural language data (you'll do this in Chapter 11), or producing novel images with a generative adversarial network (as in Chapter 12).

Reinforcement Learning

Returning to Figure 4.1, we're now well positioned to cover *reinforcement learning* problems, which are markedly different from the supervised and unsupervised varieties. As illustrated lightheartedly in Figure 4.3, reinforcement learning problems are ones that we can frame as having an *agent* take a sequence of actions within some *environment*. The agent could, for example, be a human or an algorithm playing an Atari video game, or it could be a human or an algorithm driving a car. Perhaps the primary way in which reinforcement learning problems diverge from supervised or unsupervised ones is that the actions taken by the agent influence the information that the environment provides to the agent—that is, the agent receives direct feedback on the actions it takes. In supervised or unsupervised problems, in contrast, the model never impacts the underlying data; it simply consumes it.



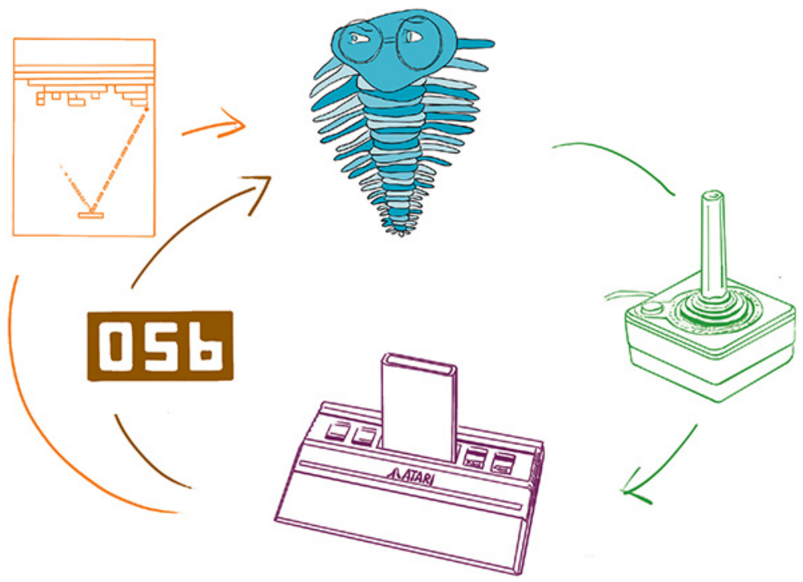


Figure 4.3 The reinforcement learning loop. The top diagram is a generalized version. The bottom diagram is specific to the example elaborated on in the text of an agent playing a video game on an Atari console. To our knowledge, trilobites can't actually play video games; we're using the trilobite as a symbolic representation of the reinforcement learning agent, which could be either a human or a machine.

Students of deep learning often have an innate desire to divide the supervised, unsupervised, and reinforcement learning paradigms into the traditional machine learning versus deep learning approaches. More specifically, they seem to want to associate supervised learning with traditional machine learning while associating unsupervised learning or reinforcement learning (or both) with deep learning. To be clear, there is no such association to be made. Both traditional machine learning and deep learning techniques can be applied to supervised, unsupervised, and reinforcement learning problems.

Let's dive a bit further into the relationship between a reinforcement learning agent and its environment by exploring some examples. In Figure 4.3, the agent is represented by an anthropomorphized trilobite, but this agent could be either human or a machine. Where the agent is playing an Atari video game:

- The possible *actions* that can be taken are the buttons that can be pressed on the video game controller. [4]

- The *environment* (the Atari console) returns information back to the agent. This information comes in two delicious flavors: *state* (the pixels on the screen that represent the current condition of the environment) and *reward* (the point score in the game, which is what the agent is endeavoring to maximize via gameplay).

- If the agent is playing Pac-Man, then selecting the action of pressing the "up" button results in the environment returning an updated state where the pixels representing the video game character on the screen have moved upward. Prior to playing any of the game, a typical reinforcement learning algorithm would not even have knowledge of this simple relationship between the "up" button and the Pac-Man character moving upward; everything is learned from the ground up via trial and error.

- If the agent selects an action that causes Pac-Man to cross paths with a pair of delectable cherries, then the environment will return a *positive reward*: an increase in points. On the other hand, if the agent selects an action that causes Pac-Man to cross paths with a spooky ghost, then the environment will return a *negative reward*: a decrease in points.

4 . We're not aware of video game-playing algorithms that literally press the buttons on the game console's controllers. They would typically interact with a video game directly via a software-based emulation. We go through the most popular open-source packages for doing this at the end of the chapter.

In a second example, where the agent is driving a car,

- The available *actions* are much broader and richer than for Pac-Man. The agent can adjust the steering column, the accelerator, and the brakes to varying degrees ranging from subtle to dramatic.

- The *environment* in this case is the real world, consisting of roads, traffic, pedestrians, trees, sky, and so on. The *state* then is the condition of the vehicle's surroundings, as perceived by a human agent's eyes and ears, or by an autonomous vehicle's cameras and lidar. [5]

- The *reward*, in the case of an algorithm, could be programmed to be *positive* for, say, every meter of distance traveled toward a destination; it could be somewhat *negative* for minor traffic infractions, and severely negative in the event of a collision.

5 . The laser-based equivalent of radar.

DEEP REINFORCEMENT LEARNING

At long last, we reach the *deep reinforcement learning* section near the center of the Venn diagram in Figure 4.3. A reinforcement learning algorithm earns its “deep” prefix when an artificial neural network is involved in it, such as to learn what actions to take when presented with a given state from the environment in order to have a high probability of obtaining a positive reward.[6] As you’ll see in the examples coming up in the next section, the marriage of deep learning and reinforcement learning approaches has proved a prosperous one. This is because:

- Deep neural networks excel at processing the complex sensory input provided by real environments or advanced, simulated environments in order to distill relevant signals from a cacophony of incoming data. This is analogous to the functionality of the biological neurons of your brain’s visual and auditory cortexes, which receive input from the eyes and ears, respectively.

- Reinforcement learning algorithms, meanwhile, shine at selecting an appropriate action from a vast scope of possibilities.

6 . Earlier in this chapter (see Figure 4.2), we indicate that the “deep learning” moniker applies to an artificial neural network that has at least three hidden layers. While in general this is the case, when used by the reinforcement learning community, the term “deep reinforcement learning” may be used even if the artificial neural network involved in the model is shallow, that is, composed of as few as one or two hidden layers.

Taken together, deep learning and reinforcement learning are a powerful problem-solving combination. Increasingly complex problems tend to require increasingly large datasets for deep reinforcement learning agents to wade through vast noise as well as vast randomness in order to discover an effective policy for what actions it should take in a given circumstance. Because many reinforcement learning problems take place in a simulated environment, obtaining a sufficient amount of data is often not a problem: The agent can simply be trained on further rounds of simulations.

Although the theoretical foundations for deep reinforcement learning have been around for a couple of decades,[7] as with AlexNet for vanilla deep learning (Figure 1.17), deep reinforcement learning has in the past few years benefited from a confluence of three tail winds:

7 . Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the Association for Computing Machinery, 38*, 58–68.

1. Exponentially larger datasets and much richer simulated environments

2. Parallel computing across many graphics processing units (GPUs) to model efficiently with large datasets as well as the breadth of associated possible states and possible actions

3. A research ecosystem that bridges academia and industry, producing a quickly developing body of new ideas on deep neural networks in general as well as on deep reinforcement learning algorithms in particular, to, for example, identify optimal actions across a wide variety of noisy states

VIDEO GAMES

Many readers of this book recall learning a new video game as a child. Perhaps while at an arcade or staring at the family's heavy cathode-ray-tube television set, you quickly became aware that missing the ball in Pong or Breakout was an unproductive move. You processed the visual information on the screen and, yearning for a score in excess of your friends', devised strategies to manipulate the controller effectively and achieve this aim. In recent years, researchers at a firm called DeepMind have been producing software that likewise learns how to play classic Atari games.

DeepMind was a British technology startup founded by Demis Hassabis (Figure 4.4), Shane Legg, and Mustafa Suleyman in London in 2010. Their stated mission was to "solve intelligence," which is to say they were interested in extending the field of AI by developing increasingly general-purpose learning algorithms. One of their early contributions was the introduction of deep Q-learning networks (DQNs; noted within Figure 4.1). Via this approach, a single model architecture was able to learn to play multiple Atari 2600 games well—from scratch, simply through trial and error.

In 2013, Volodymyr Mnih [8] and his DeepMind colleagues published [9] an article on their DQN agent, a deep reinforcement learning approach that you will come to understand intimately when you construct a variant of it yourself line by line in Chapter 13. Their agent received raw pixel values from its *environment*, a video game emulator,[10] as its *state* information—akin to the way human players of Atari games view a TV screen. In order to efficiently process this information, Mnih et al.'s DQN included a convolutional neural network (CNN), a common tactic for any deep reinforcement learning model that is fed visual data (this is why we elected to overlap "Deep RL" somewhat with "Machine Vision" in Figure 4.1). The handling of the flood of visual input from Atari games (in this case, a little over two million pixels per second) underscores how well suited deep learning in general is to filtering out pertinent features from noise. Further, playing Atari games within an emulator is a problem that is well suited to deep reinforcement learning in particular: While they provide a rich set of possible actions that are engineered to be challenging to master, there is thankfully no finite limit on the amount of training data available because the agent can engage in endless rounds of play.

8 . Mnih obtained his doctorate at the University of Toronto under the supervision of Geoff Hinton (Figure 1.16).

9 . Mnih, V., et al. (2013). Playing Atari with deep reinforcement learning. *arXiv: 1312.5602*.

10. Bellemare, M., et al. (2012). The arcade learning environment: An evaluation platform for general agents. *arXiv: 1207.4708*.

Figure 4.4 Demis Hassabis cofounded DeepMind in 2010 after completing his PhD in cognitive neuroscience at University College London.

During training, the DeepMind DQN was not provided any hints or strategies; it was provided only with state (screen pixels), reward (its point score, which it is programmed to maximize), and the range of possible actions (game-controller buttons) available in a given Atari game. The model was not altered for specific games, and yet it was able to outperform existing machine learning approaches in six of the seven games Mnih and his coworkers tested it on, even surpassing the performance of expert human players on three. Perhaps influenced by this conspicuous progress, Google acquired DeepMind in 2014 for the equivalent of half a billion U.S. dollars.

In a follow-up paper published in the distinguished journal *Nature*, Mnih and his teammates at now-Google DeepMind assessed their DQN algorithm across 49 Atari games.[11] The results are shown in Figure 4.5: It outperformed other machine learning approaches on all but three of the games (94 percent of them), and, astonishingly, it scored above human level on the majority of them (59 percent).[12]

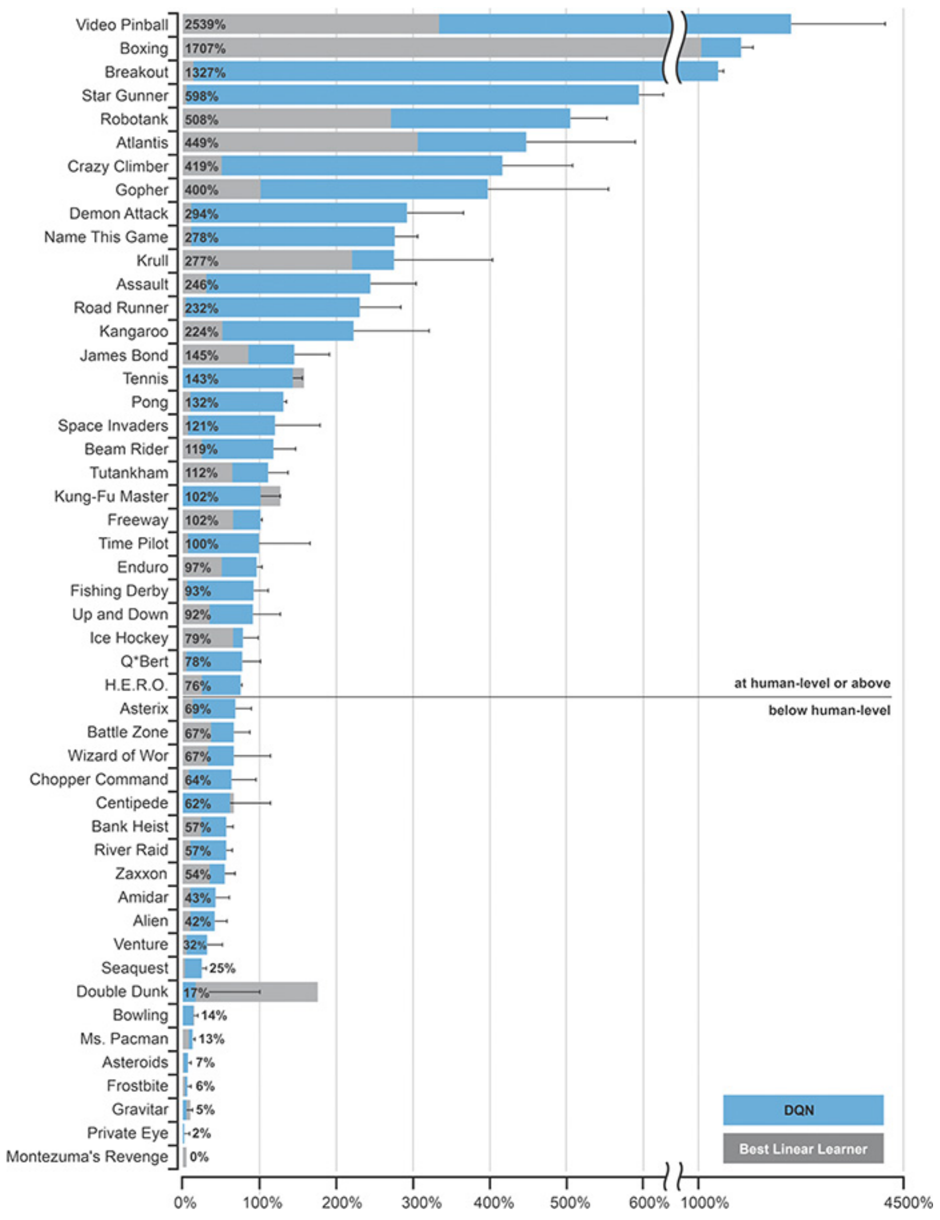


Figure 4.5 The normalized performance scores of Mnih and colleagues' (2015) DQN relative to a professional game tester: Zero percent represents random play, and 100% represents the pro's best performance. The horizontal line represents the authors' defined threshold of "human-level" play: the 75th percentile of professionals' scores.

11. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–33.

12. You can be entertained by watching the Google DeepMind DQN learn to master Space Invaders and Pong here: `bit.ly/DQNpong`.

BOARD GAMES

It might sound sensible that board games would serve as a logical prelude to video games given their analog nature and their chronological head start; however, the use of software emulators provided a simple and easy way to interact with video games digitally. Instead, the availability of these emulation tools provided the means, and so the principal advances in modern deep reinforcement learning initially took place in the realm of video games. Additionally, relative to Atari games, the complexity of some classical board games is much greater. There are myriad strategies and long-plays associated with chess expertise that are not readily apparent in Pac-Man or Space Invaders, for example. In this section, we provide an overview of how deep reinforcement learning strategies mastered the board games Go, chess, and shogi despite the data-availability and computational-complexity head winds.

AlphaGo

Invented several millennia ago in China, Go (illustrated in Figure 4.6) is a ubiquitous two-player strategy board game in Asia. The game has a simple set of rules based around the idea of capturing one's opponents' pieces (called *stones*) by encircling them with one's own.[13] This uncomplicated premise belies intricacy in practice, however. The larger board and the larger set of possible moves per turn make the game much more complex than, say, chess, for which we've had algorithms that can defeat the best human players for two decades.[14] There are a touch more than 2×10^{170} possible legal board positions in Go, which is far more than the number of atoms in the universe[15] and about a *googol* (10^{100}) more complex than chess.

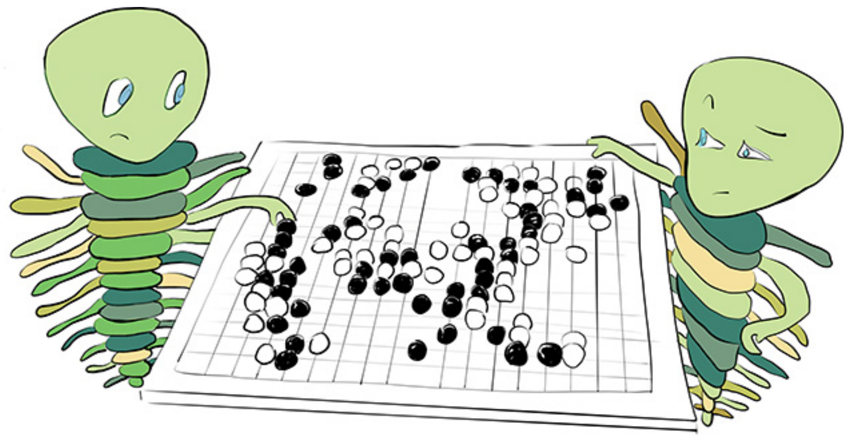


Figure 4.6 The Go board game. One player uses the white stones while the other uses the black stones. The objective is to encircle the stones of your opponent, thereby capturing them.

13. Indeed, Go in Chinese translates literally to "encirclement board game."

14. IBM's Deep Blue defeated Garry Kasparov, arguably the world's greatest-ever chess player, in 1997. More on that storied match coming up shortly in this section.

15. There are an estimated 10^{80} atoms in the observable universe.

An algorithm called *Monte Carlo tree search* (MCTS) can be employed to play uncomplicated games competently. In its purest form, MCTS involves selecting random moves[16] until the end of gameplay. By repeating this many times, moves that tended to lead to victorious game outcomes can be weighted as favorable options. Because of the extreme complexity and sheer number of possibilities within sophisticated games like Go, pure MCTS approach is impractical: There are simply too many options to search through and evaluate. Instead of pure MCTS, an alternative approach involves MCTS applied to a much more finite subset of actions that were curated by, for example, an established policy of optimal play. This curated approach has proved sufficient for defeating amateur human Go players but is uncompetitive against professionals. To bridge the gap from amateur- to professional-level capability, David Silver (Figure 4.7) and his colleagues at Google DeepMind devised a program called AlphaGo that combines MCTS with both supervised learning and deep reinforcement learning.[17]

Figure 4.7 David Silver is a Cambridge- and Alberta-educated researcher at Google DeepMind. He has been instrumental in combining the deep learning and reinforcement learning paradigms.

16. Hence “Monte Carlo”: The casino-dense district of Monaco evokes imagery of random outcomes.

17. Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–9.

Silver et al. (2016) used supervised learning on a historical database of expert human Go moves to establish something called a *policy network*, which provides a shortlist of possible moves for a given situation. Subsequently, this policy network was refined via *self-play* deep reinforcement learning, wherein both opponents are Go-playing agents of a comparable skill level. Through this self-play, the agent iteratively improves upon itself, and whenever it improves, it is pitted against its now-improved self, producing a positive-feedback loop of continuous advancement. Finally, the cherry atop the AlphaGo algorithm: a so-called *value network* that predicts the winner of the self-play games, thereby evaluating positions on the board and learning to identify strong moves. The combination of these policy and value networks (more on both of these in Chapter 13) reduces the breadth of search space for the MCTS.

AlphaGo was able to win the vast majority of games it played against other computer-based Go programs. Perhaps most strikingly, AlphaGo was also able to defeat Fan Hui, the then-reigning

European Go champion, five games to zero. This marked the first time a computer defeated a professional human player in a full play of the game. As exemplified by the Elo ratings[18] in Figure 4.8, AlphaGo performed at or above the level of the best players in the world.

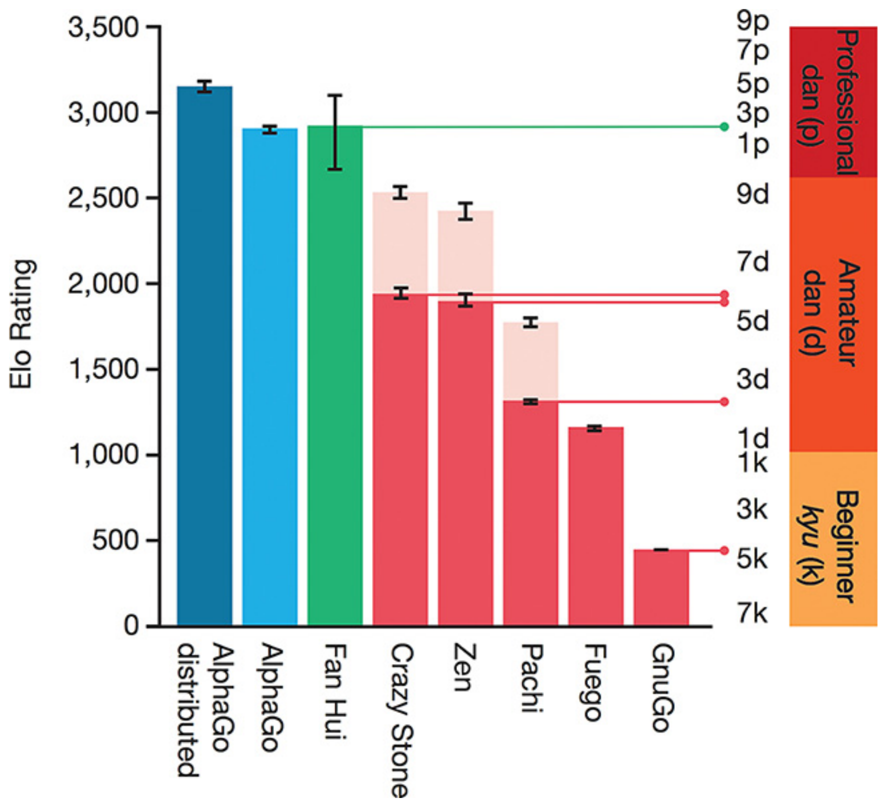


Figure 4.8 The Elo score of AlphaGo (blue) relative to Fan Hui (green) and several Go programs (red). The approximate human rank is shown on the right.

18. Elo ratings enable the skill level of human and artificial game players alike to be compared. Derived from calculations of head-to-head wins and losses, an individual with a higher Elo score is more likely to win a game against an opponent with a lower score. The larger the score gap between the two players, the greater the probability that the player with the higher score will win.

Following this success, AlphaGo was famously matched against Lee Sedol in March 2016 in Seoul, South Korea. Sedol has 18 world titles and is considered one of the all-time great players. The five-game match was broadcast and viewed live by 200 million people. AlphaGo won the match 4-1, launching DeepMind, Go, and the artificially intelligent future into the public imagination.[19]

19. There is an outstanding documentary on this Sedol match that gave us chills: Kohs, G. (2017). *AlphaGo*. United States: Moxie Pictures & Reel As Dirt.

AlphaGo Zero

Following AlphaGo, the folks at DeepMind took their work further and created a second-generation Go player: AlphaGo Zero. Recall that AlphaGo was initially trained in a supervised manner; that is, expert human moves were used to train the network first, and thereafter the network learned by reinforcement learning through self-play. Although this is a nifty approach, it doesn't exactly "solve intelligence" as DeepMind's founders would have liked. A better approximation of general intelligence would be a network that could learn to play Go in a completely *de novo* setting—where the network is not supplied with any human input or domain knowledge, but improves by deep reinforcement learning alone. Enter AlphaGo Zero.

As we've alluded to before, the game of Go requires sophisticated look-ahead capabilities through vast search spaces. That is, there are so many *possible* moves and such a tiny fraction of them are *good* moves in the short- and longplay of the game that performing a search for the optimal move, keeping the likely future state of the game in mind, becomes exceedingly complex and computationally impractical. It is for this reason that it was thought that Go would be a final frontier for machine intelligence; indeed, it was thought that the achievements of AlphaGo in 2016 were a decade or more away.

Working off the momentum from the AlphaGo-Sedol match in Seoul, researchers at DeepMind created AlphaGo Zero, which learns to play Go far beyond the level of the original AlphaGo—while being revolutionary in several ways.[20] First and foremost, it is trained without any data from human gameplay. That means it learns purely by trial and error. Second, it uses only the stones on the board as inputs. Contrastingly, AlphaGo had received 15 supplementary, human-engineered features, which provided the algorithm key hints such as how many turns since a move was played or how many opponent stones would be captured. Third, a single (deep) neural network was used to evaluate the board and decide on a next move, rather than separate policy and value networks (as mentioned in the sidebar on page 61; more on these coming in Chapter 13). Finally, the tree search is simpler and relies on the neural network to evaluate positions and possible moves.

20. Silver, D., et al. (2016). Mastering the game of Go without human knowledge. *Nature* 550, 354–359.

AlphaGo Zero played almost five million games of self-play over three days, taking an estimated 0.4s per move to "think." Within 36 hours, it had begun to outperform the model that beat Lee Sedol in Seoul (retroactively termed AlphaGo Lee), which—in stark contrast—took several months to train. At the 72-hour mark, the model was pitted against AlphaGo Lee in match conditions, where it handily won every single one of 100 games. Even more remarkable is that AlphaGo Zero achieved this on a single machine with four tensor processing units (TPUs)[21] whereas AlphaGo Lee was distributed over multiple machines and used 48 TPUs. (AlphaGo Fan, which beat Fan Hui, was distributed over 176 GPUs!) In Figure 4.9, the Elo score for AlphaGo Zero is shown over days of training time and compared to the scores for AlphaGo Master[22] and AlphaGo Lee. Shown on the right are the absolute Elo scores for a variety of iterations of AlphaGo and some other Go programs. AlphaGo Zero is far and away the superior model.

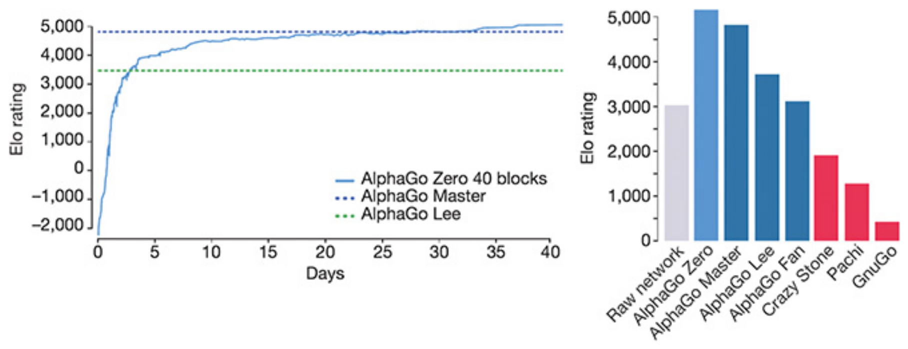


Figure 4.9 Comparing Elo scores between AlphaGo Zero and other AlphaGo variations or other Go programs. In the left-hand plot, the comparison is over days of AlphaGo Zero training.

21. Google built custom processor units for training neural networks, known as tensor processing units (TPUs). They took the existing architecture of a GPU and specifically optimized it for performing calculations that predominate the training of neural network models. At the time of writing, TPUs were accessible to the public via the Google Cloud Platform only.

22. AlphaGo Master is a hybrid between AlphaGo Lee and AlphaGo Zero; however, it uses the extra input features enjoyed by AlphaGo Lee and initializes training in a supervised manner. AlphaGo Master famously played online anonymously in January 2017 under the pseudonyms *Master* and *Magister*. It won all 60 of the games it played against some of the world's strongest Go players.

A startling discovery that emerged from this research was that the nature of the gameplay by AlphaGo Zero is qualitatively different from that of human players and (the human gameplay-trained) AlphaGo Lee. AlphaGo Zero began with random play but quickly learned professional *joseki*—corner sequences that are considered heuristics of distinguished play. After further training, however, the mature model tended to prefer novel *joseki* that were previously unknown to humankind. AlphaGo Zero did spontaneously learn a whole range of classical Go moves, implying a pragmatic alignment with these techniques. However, the model did this in an original manner: It did not learn the concept of *shicho* (ladder sequences), for example, until much later in its training, whereas this is one of the first concepts taught to novice human players. The authors additionally trained another iteration of the model with human gameplay data. This supervised model performed better initially; however, it began to succumb to the data-free model within the first 24 hours of training and ultimately achieved a lower Elo score. Together, these results suggest that the data-free, self-learned model has a style of play distinct from that of human players—a dominating style that the supervised model fails to develop.

AlphaZero

Having trounced the Go community, the team at DeepMind shifted their focus to general game-playing neural networks. Although AlphaGo Zero is adept at playing Go, they wondered if a comparable network could learn to play multiple games expertly. To put this to the test, they added two new games to their repertoire: chess and shogi.[23]

23. Silver, D., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv:1712.01815*.

Most readers are likely familiar with the game of chess, and shogi—referred to by some as Japanese chess—is similar. Both games are two-player strategy games, both take place on a grid-format board, both culminate in a checkmate of the opponent's king, and both consist of a range of pieces with different moving abilities. Shogi, however, is significantly more complex than chess, with a larger board size (9×9, relative to 8×8 in chess) and the fact that opponent pieces can be replaced anywhere on the board after their capture.

Historically, artificial intelligence has had a rich interaction with the game of chess. Over several decades, chess-playing computer programs have been developed extensively. The most famous is Deep Blue, conceived by IBM, which went on to beat the world champion Garry Kasparov in 1997.[24] It was heavily reliant on brute-force computing power[25] to execute complex searches through possible moves, and combined this with handcrafted features and domain-specific adaptations. Deep Blue was fine-tuned by analyzing thousands of master games (it was a supervised learning system!) and it was even tweaked between games.[26]

24. Deep Blue lost its first match against Kasparov in 1996, and after significant upgrades went on to narrowly beat Kasparov in 1997. This was not the total domination of man by machine that AI proponents might have hoped for.

25. Deep Blue was the planet's 259th most powerful supercomputer at the time of the match against Kasparov.

26. This tweaking was a point of contention between IBM and Kasparov after his loss in 1997. IBM refused to release the program's logs and dismantled Deep Blue. Their computer system never received an official chess ranking, because it played so few games against rated chess masters.

Although Deep Blue was an achievement two decades ago, the system was not generalizable; it could not perform any task other than chess. After AlphaGo Zero demonstrated that the game of Go could be learned by a neural network from first principles alone, given nothing but the board and the rules of the game, Silver and his DeepMind colleagues set out to devise a generalist neural network, a *single network architecture* that could dominate not only at Go but also at other board games.

Compared to Go, chess and shogi present pronounced obstacles. The rules of the games are position dependent (pieces can move differently based on where they are on the board) and asymmetrical (some pieces can move only in one direction).[27] Long-range actions are possible (such as the queen moving across the entire board), and games can result in draws.

27. This makes expanding the training data via synthetic augmentation—an approach used copiously for AlphaGo—more challenging.

AlphaZero feeds the board positions into a neural network and outputs a vector of move probabilities for each possible action, as well as a scalar[28] outcome value for that move. The network learns the parameters for these move probabilities and outcomes entirely from self-play deep reinforcement learning, as AlphaGo Zero did. An MCTS is then performed on the reduced space guided by these probabilities, returning a refined vector of probabilities over the possible moves. Whereas AlphaGo Zero optimizes the probability of winning (Go is a binary win/loss game), AlphaZero instead optimizes for the expected outcome. During self-play, AlphaGo Zero retains the best *player* to date and evaluates updated versions of itself against that player, continually replacing the player with the next best version. AlphaZero, in contrast, maintains a single network and at any given time is playing against the latest version of itself. AlphaZero was trained to play each of chess, shogi, and Go for a mere 24 hours. There were no game-specific modifications, with the exception of a manually configured parameter that regulates how frequently the model takes random, exploratory moves; this was scaled to the number of legal moves in each game.[29]

28. A single value.

29. This manually configured exploration parameter is called *epsilon*. It is detailed in Chapter 13.

Across 100 competitive games, AlphaZero did not lose a single one against the 2016 Top Chess Engine Championship world champion Stockfish. In shogi, the Computer Shogi Association world champion, Elmo, managed to beat AlphaZero only eight times in 100 games. Its perhaps most worthy opponent, AlphaGo Zero, was able to defeat AlphaZero in 40 of their 100 games. Figure 4.10 shows the Elo scores for AlphaZero relative to these three adversaries.

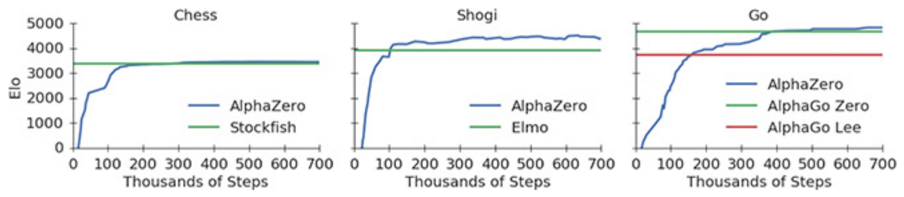


Figure 4.10 Comparing Elo scores between AlphaZero and each of its opponents in chess, shogi, and Go. AlphaZero rapidly outperformed all three opponents.

Not only was AlphaZero superior, it was also efficient. AlphaZero's Elo score exceeded its greatest foes' after only two, four, and eight hours of training for shogi, chess, and Go, respectively. This is a sensationally rapid rate of learning, considering that in the case of Elmo and Stockfish, these computer programs represent the culmination of decades of research and fine-tuning in a focused, domain-specific manner. The generalizable AlphaZero algorithm is able to play all three games with aplomb: Simply switching out learned weights from otherwise identical neural network architectures imbues each with the same skills that have taken *years* to develop by other means. These results demonstrate that deep reinforcement learning is a strikingly powerful approach for developing general expert gameplay in an undirected fashion.

MANIPULATION OF OBJECTS

As this chapter's title might suggest, thus far we've centered our coverage of deep reinforcement learning on its game-playing applications. Although games offer a hot testbed for exploring the generalization of machine intelligence, in this section we spend a few moments expounding on practical, real-world applications of deep reinforcement learning.

One real-world example we mention earlier in this chapter is autonomous vehicles. As an additional example, here we provide an overview of research by Sergey Levine, Chelsea Finn (Figure 4.11), and labmates at the University of California, Berkeley.[30] These researchers trained a robot to perform a number of motor skills that require complex visual understanding and depth perception, such as screwing the cap back onto a bottle, removing a nail with a toy hammer, placing a hanger on a rack, and inserting a cube in a shape-fitting game (Figure 4.12).

Figure 4.11 Chelsea Finn is a doctoral candidate at the University of California, Berkeley, in its AI Research Lab.

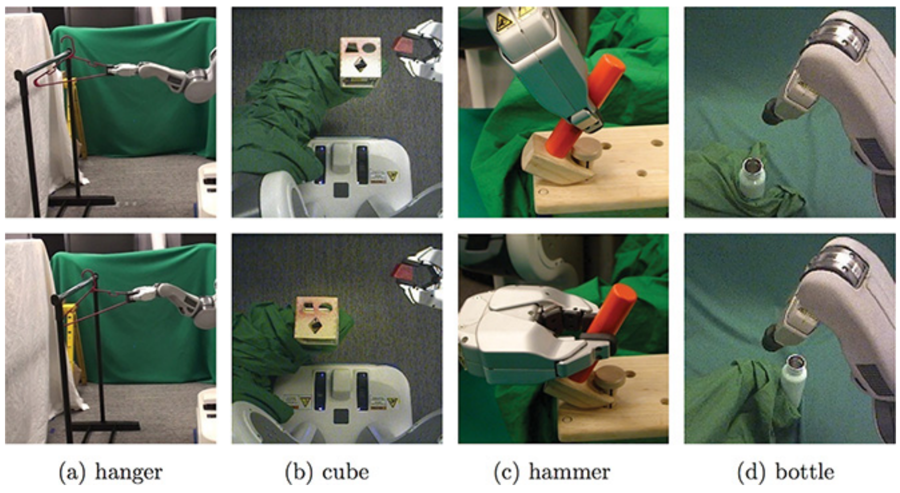


Figure 4.12 Sample images from Levine, Finn, et al. (2016) exhibiting various object-manipulation actions the robot was trained to perform

30. Levine, S., Finn, C., et al. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17, 1–40.

Levine, Finn, and colleagues’ algorithm maps raw visual input directly to the movement of the motors in the robot’s arm. Their policy network was a seven-layer-deep convolutional neural network (CNN) consisting of fewer than 100,000 artificial neurons—a minuscule amount in deep learning terms, as you’ll see when you train orders-of-magnitude larger networks later in this book. Although it would be tricky to elaborate further on this approach before we delve much into artificial-neural-network theory (in Part II, which is just around the corner), there are three take-away points we’d like to highlight on this elegant practical application of deep reinforcement learning. First, it is an “end-to-end” deep learning model in the sense that the model takes in raw images (pixels) as inputs and then outputs directly to the robot’s motors. Second, the model generalizes neatly to a broad range of unique object-manipulation tasks. Third, it is an example of the *policy gradient* family of deep reinforcement learning approaches, rounding out the terms featured in the Venn diagram in Figure 4.1. Policy gradient methods are distinct from the DQN approach that is the focus of Chapter 13, but we touch on them then too.

POPULAR DEEP REINFORCEMENT LEARNING ENVIRONMENTS

Over the past few sections, we talk a fair bit about software emulation of environments in which to train reinforcement learning models. This area of development is crucial to the ongoing progression of reinforcement learning; without environments in which our agents can play and explore (and gather data!), there would be no training of models. Here we introduce the three most popular environments, discussing their high-level attributes.

OpenAI Gym

OpenAI Gym[31] is developed by the nonprofit AI research company OpenAI.[32] The mission of OpenAI is to advance artificial general intelligence (more on that in the next section!) in a safe and equitable manner. To that end, the researchers at OpenAI have produced and open-sourced a number of tools for AI research, including the OpenAI Gym. This toolkit is designed to provide an interface for training reinforcement learning models, be they deep or otherwise.

31. github.com/openai/gym (http://github.com/openai/gym)

32. openai.com (http://openai.com)

As captured in Figure 4.13, the Gym features a diverse array of environments, including a number of Atari 2600 games,[33] multiple robotics simulators, a few simple text-based algorithmic games, and several robotics simulations using the MuJoCo physics engine.[34] In Chapter 13, you’ll install OpenAI Gym in a single line of code and then employ an environment it provides to train the DQN agent that you build. OpenAI Gym is written in Python and is compatible with any deep learning computation library, including TensorFlow and PyTorch (we discuss the various deep learning libraries in Chapter 14; these are two particularly popular ones).

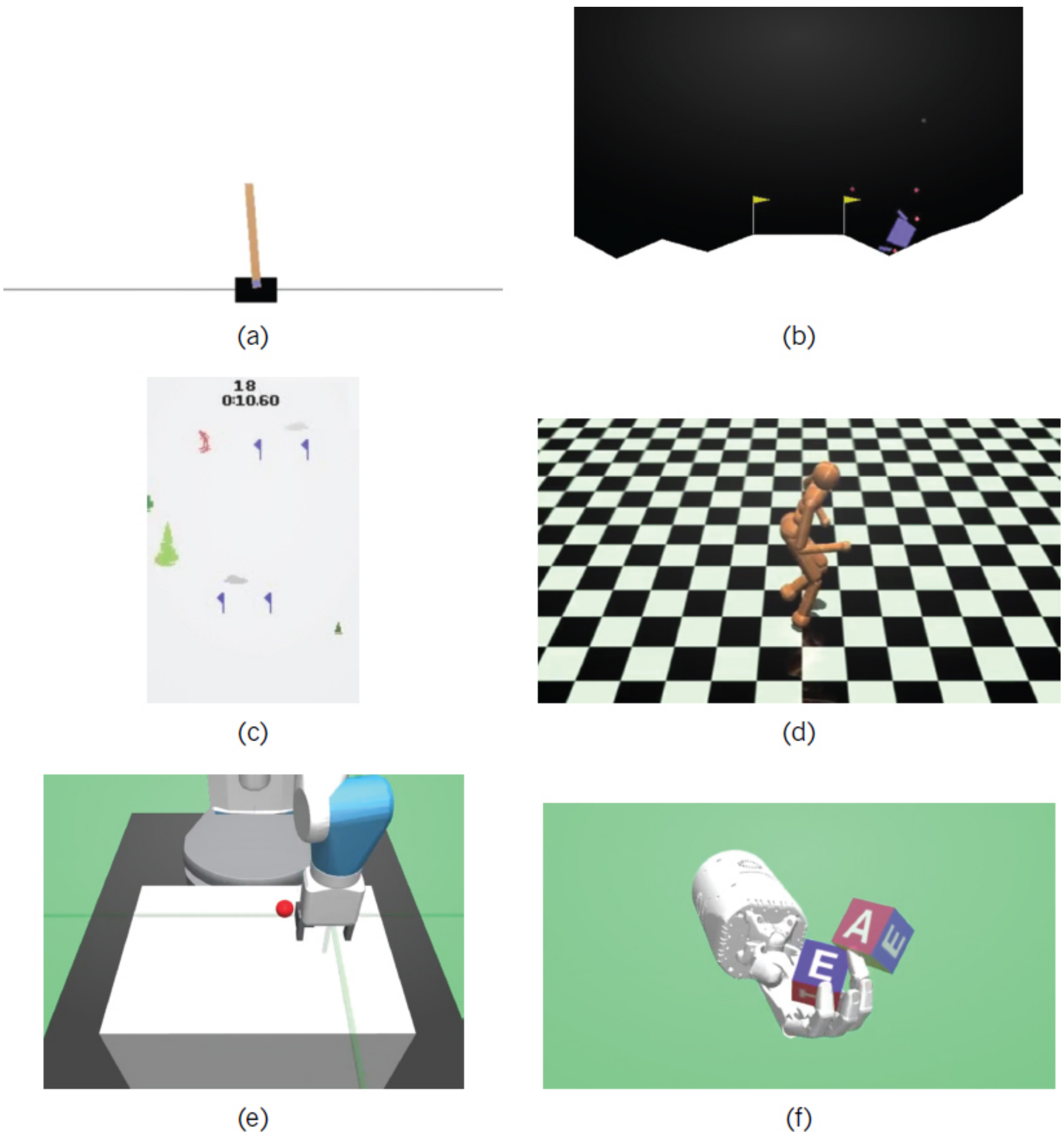


Figure 4.13 A sampling of OpenAI Gym environments: (a) CartPole, a classic control-theory problem; (b) LunarLander, a continuous-control task run inside a two-dimensional simulation; (c) Skiing, an Atari 2600 game; (d) Humanoid, a three-dimensional MuJuCo physics engine simulation of a bipedal person; (e) FetchPickAndPlace, one of several available simulations of real-world robot arms, in this case involving one called Fetch, with the goal of grasping a block and placing it in a target location; and (f) HandManipulateBlock, another practical simulation of a robotic arm, the Shadow Dexterous Hand.

33. OpenAI Gym uses the Arcade Learning Environment to emulate Atari 2600 games. This same framework is used in the Mnih et al. (2013) paper described in the "Video Games" section. You can find the framework yourself at github.com/mgbellemare/Arcade-Learning-Environment (http://github.com/mgbellemare/Arcade-Learning-Environment).

34. MuJoCo is an abbreviation of Multi-Joint dynamics with Contact. It is a physics engine that was developed by Emo Todorov for Roboti LLC.

DeepMind Lab

DeepMind Lab[35] is another RL environment, this time from the developers at Google DeepMind (although they point out that DeepMind Lab is *not* an official Google product). As can be seen in Figure 4.14, the environment is built on top of id Software's Quake III Arena[36] and provides a sci-fi inspired three-dimensional world in which agents can explore. The agent experiences the environment from the first-person perspective, which is distinct from the Atari emulators available via OpenAI Gym.

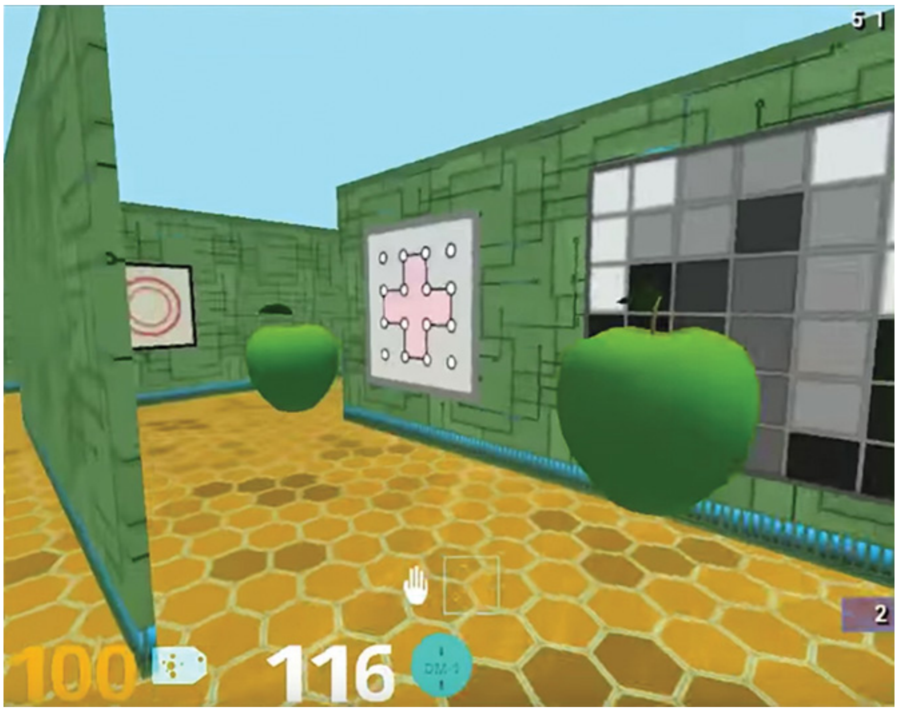


Figure 4.14 A DeepMind Lab environment, in which positive-reward points are awarded for capturing scrumptious green apples

35. Beattie, C. et al. (2016). DeepMind Lab. *arXiv:1612.03801*.

36. Quake III Arena. (1999). United States: id Software. github.com/id-Software/Quake-III-Arena (http://github.com/id-Software/Quake-III-Arena)

There are a variety of levels available, which can be roughly divided into four categories:

1. Fruit-gathering levels, where the agent simply tries to find and collect rewards (apples and melons) while avoiding penalties (lemons).

2. Navigation levels with a static map, where the agent is tasked with finding a goal and remembering the layout of the map. The agent can either be randomly placed within a map at the start of each episode while the goal remains stationary, an arrangement that tests initial exploration followed by a reliance on memory to repeatedly find the goal; or the agent can start in the same place while the goal is moved for every episode, testing the agent’s ability to explore.

3. Navigation levels with random maps, where the agent is required to explore a novel map in each episode, find the goal, and then repeatedly return to the goal as many times as possible within a time limit.

4. Laser-tag levels, where the agent is rewarded for hunting and attacking bots in an array of different scenes.

Installation of DeepMind Lab is not as straightforward as OpenAI Gym,[37] but it provides a rich, dynamic first-person environment in which to train agents, and the levels provide complex scenarios involving navigation, memory, strategy, planning, and fine-motor skills. These challenging environments can test the limits of what is tractable with contemporary deep reinforcement learning.

37. First the Github repository (github.com/deepmind/lab (http://github.com/deepmind/lab)) is cloned, and then the software must be built using Bazel (`bit.ly/installB`). The DeepMind Lab repository provides detailed instructions (`bit.ly/buildDML`).

Unity ML-Agents

Unity is a sophisticated engine for two- and three-dimensional video games and digital simulations. Given the game-playing proficiency of reinforcement learning algorithms we chronicle earlier in this chapter, it may come as no surprise that the makers of a popular game engine are also in the business of providing environments to incorporate reinforcement learning into video games. The Unity ML-Agents plug-in[38] enables reinforcement learning models to be trained within Unity-based video games or simulations and, perhaps more fitting with the purpose of Unity itself, allows reinforcement learning models to guide the actions of agents within the game.

38. github.com/Unity-Technologies/ml-agents (http://github.com/Unity-Technologies/ml-agents)

As with DeepMind Lab, installation of Unity ML-Agents is not a one-liner.[39]

39. It requires the user to first install Unity (for download and installation instructions, see store.unity.com/download (http://store.unity.com/download)) and then clone the Github repository. Full instructions are available at the Unity ML-Agents Github repository (`bit.ly/MLagents`).

THREE CATEGORIES OF AI

Of all deep learning topics, deep reinforcement learning is perhaps the one most closely tied to the popular perception of artificial intelligence as a system for replicating the cognitive, decision-making capacity of humans. In light of that, to wrap up this chapter, in this section we introduce three categories of AI.

Artificial Narrow Intelligence

Artificial narrow intelligence (ANI) is machine expertise at a specific task. Many diverse examples of ANI exist today, and we’ve mentioned plenty already in this book, such as the visual recognition of objects, real-time machine translation between natural languages, automated financial-trading systems, AlphaZero, and self-driving cars.

Artificial General Intelligence

Artificial general intelligence (AGI) would involve a single algorithm that could perform well at all of the tasks described in the preceding paragraph: It would be able to recognize your face, translate this book into another language, optimize your investment portfolio, beat you at Go, and drive you safely to your holiday destination. Indeed, such an algorithm would be approximately indistinguishable from the intellectual capabilities of an individual human. There are countless hurdles to overcome in order for AGI to be realized; it is challenging to approximate when it will be achieved, if it will be achieved at all. That said, AI experts are happy to wave a finger in the air and speculate on timing. In a study conducted by the philosopher Vincent Müller and the influential futurist Nick Bostrom,[40] the median estimate across hundreds of professional AI researchers is that AGI will be attained in the year 2040.

40. Müller, V., and Bostrom, N. (2014). Future progress in artificial intelligence: A survey of expert opinion. In V. Müller (Ed.), *Fundamental Issues of Artificial Intelligence*. Berlin: Springer.

Artificial Super Intelligence

Artificial super intelligence (ASI) is difficult to describe because it's properly mind-boggling. ASI would be an algorithm that is markedly more advanced than the intellectual capabilities of a human.[41] *If* AGI is possible, then ASI may be as well. Of course, there are even more hurdles on the road to ASI than to AGI, the bulk of which we can't foresee today. Citing the Müller and Bostrom survey again, however, AI experts' median estimate for the arrival of ASI is 2060, a rather hypothetical date that falls within the life-span of many earthlings alive today. In Chapter 14, at which point you'll be well-versed in deep learning both in theory and in practice, we discuss both how deep learning models could contribute to AGI as well as the present limitations associated with deep learning that would need to be bridged in order to attain AGI or, gasp, ASI.

41. In 2015, the writer and illustrator Tim Urban provided a two-part series of posts that rivetingly covers ASI and the related literature. It's available at `bit.ly/urbanAI` for you to enjoy.

SUMMARY

The chapter began with an overview relating deep learning to the broader field of artificial intelligence. We then detailed deep reinforcement learning, an approach that blends deep learning with the feedback-providing reinforcement learning paradigm. As discussed via real-world examples ranging from the board game Go to the grasping of physical objects, such deep reinforcement learning enables machines to process vast amounts of data and take sensible sequences of actions on complex tasks, associating it with popular conceptions of AI.