Perspectives on Research in Artificial Intelligence and Artificial General Intelligence Relevant to DoD

Contact: Richard Potember — rpotember@mitre.org

January 2017

JSR-16-Task-003

JASON
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102-7508
(703) 983-6997

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
January 2017		

4. TITLE AND SUBTITLE

Perspectives on Research in Artificial Intelligence and Artificial General Intelligence Relevant to DoD

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER
1316JA01

5e. TASK NUMBER
PS

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The MITRE Corporation
JASON Program Office
7515 Colshire Drive, MS T130
McLean, Virginia 22102

8. PERFORMING ORGANIZATION REPORT NUMBER

JSR-16-Task-003

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

OSD ASDR&E
Basic Research Labs
4800 Mark Center Drive
Alexandria VA

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution authorized for Public Release

13. SUPPLEMENTARY NOTES

Artificial Intelligence (AI) is conventionally, if loosely, defined as intelligence exhibited by machines. Operationally, it can be defined as those areas of R&D practiced by computer scientists who identify with one or more of the following academic sub-disciplines: Computer Vision, Natural Language Processing (NLP), Robotics (including Human-Robot Interactions), Search and Planning, Multi-agent Systems, Social Media Analysis (including Crowdsourcing), and Knowledge Representation and Reasoning (KRR). The field of Machine Learning (ML) is a foundational basis for AI. While this is not a complete list, it captures the vast majority of AI researchers.

Artificial General Intelligence (AGI) is a research area within AI, small as measured by numbers of researchers or total funding, that seeks to build machines that can successfully perform *any* task that a human might do. Where AI is oriented around specific tasks, AGI seeks general cognitive abilities. On account of this ambitious goal, AGI has high visibility, disproportionate to its size or present level of success, among futurists, science fiction writers, and the public.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Robin Staffin
Unclassified	Unclassified	Unclassified	UL		**19b. TELEPHONE NUMBER** *(include area code)* 571-372-6460

Contents

1 EXECUTIVE SUMMARY

1.1 Background

Artificial Intelligence (AI) is conventionally, if loosely, defined as intelligence exhibited by machines. Operationally, it can be defined as those areas of R&D practiced by computer scientists who identify with one or more of the following academic sub-disciplines: Computer Vision, Natural Language Processing (NLP), Robotics (including Human-Robot Interactions), Search and Planning, Multi-agent Systems, Social Media Analysis (including Crowdsourcing), and Knowledge Representation and Reasoning (KRR). The field of Machine Learning (ML) is a foundational basis for AI. While this is not a complete list, it captures the vast majority of AI researchers.

Artificial General Intelligence (AGI) is a research area within AI, small as measured by numbers of researchers or total funding, that seeks to build machines that can successfully perform *any* task that a human might do. Where AI is oriented around specific tasks, AGI seeks general cognitive abilities. On account of this ambitious goal, AGI has high visibility, disproportionate to its size or present level of success, among futurists, science fiction writers, and the public.

This JASON study was sponsored by DoD/OSD/ASD(R&E). AI technologies are of great importance to DoD missions. Defense systems and platforms with varying degrees of autonomy already exist. More importantly, AI is seen as the key enabling technology (along with human-computer interactions of various kinds) of a "Third Offset Strategy" that seeks for the U.S. a unique, asymmetric advantage over near-peer adversaries.

1.2 Overview

Starting around 2010, the field of AI has been jolted by the broad and unforeseen successes of a specific, decades-old technology: multi-layer neural networks (NNs). This phase-change re-energizing of a particular area of AI is the result of two evolutionary developments that together crossed a qualitative threshold: (i) fast hardware Graphics Processor Units (GPUs) allowing the training of much larger—and especially deeper (i.e., more layers)—networks, and (ii) large labeled data sets (images, web queries, social networks, etc.) that could be used as training test-beds. This combination has given rise to the "data-driven paradigm" of Deep Learning (DL) on deep neural networks (DNNs), especially with an architecture termed Convolutional Neural Networks (CNNs).

Hardware and software concepts for DNNs have synergistically co-evolved. GPU capabilities are increasingly well matched to DNN architectures that are empirically found to be the most powerful. Special purpose hardware accelerators for DNNs exist in laboratories and proprietary industry settings and may soon become available commercially.

DL methods (including hybrid methods with other technologies) already exceed human performance in some kinds of image recognition, spoken word recognition, the game of Go (long thought to require generalized human intelligence—AGI, roughly speaking). DL has notably made self- driving vehicles practical. They are now limited less by technology than by policy.

Deep Learning, based on DNNs trained on Big Data, is a tipping point in AI, evangelized by many fervent supporters. As a "dogma", DL has these beliefs: (i) Use of DNNs, often convolutional, at scale. (ii) Flat, numerical data representations. Inputs are vectors of reals. Internal data representations are tens to hundreds of millions of real-valued activations. (iii) Desirability of training on Big Data with few hard-wired model assumptions. DL seeks to learn everything from the data, believing that "data is where truth lies". (iv) The strong belief that an approximate answer is good enough. When a solution works, use it and don't ask too many questions about *how* it works.

Nevertheless, the very real successes of the DL revolution may be overshadowing some other rapidly advancing areas in AI. The report discusses the successes of reinforcement learning (RL, which can be applied both to DL and other paradigms); graphical and Bayes models, especially with probabilistic programming languages; generative models that may allow training with much smaller data sets; and other kinds of probabilistic models such as those that have shown remarkable successes in question answering (e.g., IBM's Watson), machine translation, and robotics. While DL will certainly affect all of these fields, it is not the only or final answer. More likely, DL will become an essential building block in more complicated, hybrid AI architectures.

In a seminal paper 25 years ago, Shaw pointed out that engineering disciplines grow from an initial "craft" stage, through a "commercial" but still fundamentally empirical state, and only finally reach a stage that can be called "professional engineering" (in which the "ilities" are accounted for). AI, especially Deep Learning, is somewhere early in the second stage.

The so-called "ilities" are of particular importance to DoD applications: reliability, maintainability, accountability, verifiability, evolvability, attackability, and so forth. As a generalization, DL—in its current state of development—is weak on the "ilities". The full report discusses why, at a fundamental level, this is the case: DNNs are function approximators in very high dimensional spaces (e.g., millions of dimensions). The manifolds whose shape and extent they are attempting to approximate are almost unknowably intricate, leading to failure modes for which—currently—there is very little human intuition, and even less established engineering practice.

JASON's findings and recommendations are given in Chapter 8.

2 INTRODUCTION AND BACKGROUND

2.1 AI and AGI

Artificial Intelligence (AI), defined loosely as the ability of machines (computers) to perform tasks that humans do with their brains, has captured the American public's imagination in ways both good and bad. Smartphone and computer users readily embrace the use of AI technologies in speech recognition, Internet search, and Facebook image tagging. Movie viewers take for granted the spectacular, or sometimes subtle, digital special effects made possible by AI. Sports fans now expect to see three-dimensional reconstructions of key plays in realistic video. The video game industry is larger than the movie industry. Fully self-driving cars exist in prototype, while partial self-driving features are already widely available to consumers.

At the same time, there is a growing public suspicion of AI, not always based on fact, especially in some applications relevant to DoD missions. Humanoid robots of malign intent are a Hollywood staple—ironically, rendered on film by benign AI technologies. Lethal unmanned aerial vehicles (UAVs), at present fully under human command and control, are readily conflated in the public's mind with futuristic, fully autonomous killer robots whose human control is imagined to be tenuous and fragile. In January, 2015, an Open Letter on Artificial Intelligence was signed by luminaries including Stephen Hawking and Elon Musk, that, while recognizing AI's benefits, cautioned against dire consequences. Hawking told the BBC, "The primitive forms of artificial intelligence we already have proved very useful. But I think the development of full artificial intelligence could spell the end of the human race." Musk has described AI as "our biggest existential threat".[1]

To most computer scientists, the claimed "existential threats" posed by AI seem at best uninformed. They do not align with the most rapidly advancing current research directions of AI as a field, but rather spring from dire predictions about one small area of research within AI, Artificial General Intelligence (AGI). AGI seeks to develop machines with "generalized" human intelligence, capable of sustaining long-term goals and intent, or, more generally "perform any intellectual task that a human being can."[2] Where AI is oriented around specific tasks, AGI seeks general cognitive abilities. On account of this ambitious goal, AGI has high visibility, disproportionate to its size or present level of success. Further, as this report elaborates in subsequent sections, the breakout technologies that have put us in a "golden age" of AI, may impact AGI only modestly. In the midst of an AI revolution, there are no present signs of any corresponding revolution in AGI. On this issue, the AI100 Study Panel, a consensus effort by a broad set of prominent AI researchers, recently concluded,

> "Contrary to the more fantastic predictions for AI in the popular press, the Study Panel found no cause for concern that AI is an imminent threat to humankind. *No machines with self-sustaining long-term goals and intent have been developed, nor are they likely*

[1] http://www.telegraph.co.uk/technology/news/11342200/Top-scientists-call-for-caution-over-artificial-intelligence.html

[2] Wikipedia, at https://en.wikipedia.org/wiki/Artificial_general_intelligence . Wikipedia goes on to say that AGI is "an important topic for science fiction writers and futurists.

to be developed in the near future. Instead, increasingly useful applications of AI, with potentially profound positive impacts on our society and economy are likely to emerge between now and 2030, the period this report considers." [3] (emphasis added)

This JASON study was sponsored by the Assistant Secretary of Defense for Research and Engineering (ASD R&E) within the Office of the Secretary of Defense (OSD), Department of Defense (DoD). The study looks at AI research at the "6.1" level (that is, unclassified basic research). We were not briefed on any DoD developmental efforts or programs of record. All briefings to JASON were unclassified, and were largely from the academic community. Our specific charge is listed in Appendix A, but in general terms it is this: Aiming at a reader with a technical background—but no assumed background in computer science or artificial intelligence—describe the technologies behind the remarkable recent advances in AI, explain how they may relate (or may not relate) to hypothesized future advances in AGI, and elucidate what special role the DoD may have in the support of basic research in these areas. Appendix B lists the briefers and topics that were input to this report.

2.2 Importance of AI to DoD

That AI and—if it were to advance significantly—AGI are of importance to DoD is so self-evident that it needs little elucidation here. Weapons systems and platforms with varying degrees of autonomy exist today in all domains of modern warfare, including air, sea (surface and underwater), and ground. To cite a few out of many possible examples: Northrop Grumman's X-47B is a strike fighter-sized unmanned aircraft, part of the U.S. Navy's Unmanned Combat Air System (UCAS) Carrier Demonstration program. Currently undergoing flight testing, it is capable of aircraft carrier launch and recovery, as well as autonomous aerial refueling.[4] DARPA's Anti-Submarine Warfare Continuous Trail Unmanned Vessel (ACTUV) program recently commissioned the "Sea Hunter", a 130 ft. unmanned trimaran optimized to robustly track quiet diesel electric submarines.[5,6] The Samsung SGR-A1 is a South Korean military robot sentry designed to replace human counterparts in the Korean demilitarized zone. It is capable of challenging humans for a spoken password and, if it does not recognize the correct password in response, shooting them with either rubber bullets or lethal ammunition.[7]

It is an important point that, while these systems have some degree of "autonomy" relying on the technologies of AI, they are in no sense a step—not even a small step—towards "autonomy" in the sense of AGI, that is, the ability to set independent goals or intent. Indeed, the word "autonomy" conflates two quite different meanings, one relating to "freedom of will or action" (like humans, or as in AGI), and the other the much more prosaic ability to act in accordance with a possibly complex rule set based on possibly complex sensor input, as in the word "automatic". In using a terminology like "autonomous weapons", the DoD may, as an unintended consequence, enhance the public's confusion on this point.

---

[3] "Artificial Intelligence and Life in 2030", Report of the 2015 Study Panel (June, 2016) at https://ai100.stanford.edu/

[4] http://www.northropgrumman.com/Capabilities/x47bucas/Pages/default.aspx

[5] https://www.washingtonpost.com/news/checkpoint/wp/2016/04/08/meet-sea-hunter-the-130-foot-unmanned-vessel-the-navy-wants-to-hunt-submarines/

[6] http://www.darpa.mil/program/anti-submarine-warfare-continuous-trail-unmanned-vessel

[7] https://en.wikipedia.org/wiki/Samsung_SGR-A1

At a higher strategic level, AI is recognized by DoD as a key enabling technology in a possible Third Offset Strategy.[8] As briefed to JASON, key elements of a Third Offset Strategy include: (i) autonomous learning systems, e.g., in applications that require faster-than-human reaction times; (ii) human-machine collaborative decision making; (iii) assisted human operations, especially in combat; (iv) advanced strategies for collaboration between manned and unmanned platforms; and (v) network-enabled, autonomous weapons capable of operating in future cyber and electronic warfare environments.[9] AI, as it is currently understood as a field of "6.1" basic research, will supply enabling technologies for all of these elements. At the same time, none of these elements are dependent on future advances in AGI.

2.3 Structure and History of AI as a Field of Research

2.3.1 Sub-fields of AI Research

In the academic world, AI research is situated largely in computer science departments as one of (say) a half-dozen major subdivisions of computer science. Within AI, researchers traditionally align themselves by research and application areas. A typical list of the sub-fields of AI might be: computer vision, natural language processing, robotics (including human-robot interactions), search and planning, multi-agent systems, social media analysis (including crowdsourcing), and knowledge representation and reasoning (within which AGI would be considered a small component). That AI research is rather stove-piped has been noted by many in the field. An expert in computer vision may know rather little about natural language processing (and so forth).

Machine learning (ML) enjoys a special relationship with AI. It provides the foundational mathematical and statistical algorithms that are used in AI's application areas. If we take perceptrons and expert systems (not exactly ML, but see Section 2.3.2) as exemplifying a "pre-modern" era in ML, then we might take the "dawn of the modern" era to be exemplified by:

- Gaussian mixture models
- k-means clustering
- hidden Markov models (HMMs).

Firmly in the "modern era" as well-understood technologies are

- support vector machines
- kernel methods
- ensemble methods such as "random forest"
- regularization methods based on Bayes priors (allowing sensible models with more parameters than data)
- hierarchical Bayes models

[8] DEPSECDEF, http://www.defense.gov/News/Speeches/Speech-View/Article/606641/the-third-us-offset-strategy-and-its-implications-for-partners-and-allies . The "First Offset Strategy" refers to the development of nuclear weapons, the "Second Offset Strategy" to precision guided munitions.
[9] Briefing by ASD (R&E) Steven Welby.

Continuing the metaphor, we are now, just since about the year 2000, in a "post-modern" era characterized by very rapid advances in, arguably, just a small number of areas,

- deep neural networks (DNNs), including convolutional neural networks (CNNs), when combined with big data (yielding so-called Deep Learning or DL)
- graphical Bayes models, including statistical inference on large Bayes nets
- reinforcement learning (RL)

Indeed, many argue (including, for example, the AI100 study panel) that just the first of these, Deep Learning, has displaced most traditional paradigms of AI research. Rarely does any field of science advance as far and as fast as AI has advanced in the last half-dozen years, thanks mostly to DL. We will spend much of this report discussing this phenomenon in some technical detail.

2.3.2 AI in Historical Context

As a field of research, AI traces its history back to the dawn of computer science, in the 1950s. The term "AI" itself was coined in 1956.[10] In the 1960s, "perceptrons" (see Section 3.1for technical details) were taken to exemplify the possibility that a machine could "learn" from data. By 1969, however, it was understood[11] that the perceptron model was not the desired "universal function approximator" that machine learning required. In particular, as essentially linear discriminators, perceptrons could not learn a task as simple as how compute logical exclusive-or (XOR). Machine learning using networks, and to some extent the field of AI as a whole, went into a decade-long decline.

In the 1980s, a resurgence of interest in AI was fueled by so-called *expert systems*, formal schemes for capturing (through interviews) and converting to actionable rules (largely if-then rules) the field-specific knowledge of human experts. It is said that, by the end of the 1980s, two thirds of Fortune 500 companies were applying expert system technology to their daily business activities.[12] Even so, as a research area, the sub-field of expert systems petered out. There was nothing "wrong" about it; but its useful initial results were not followed by significant further advances.

The 1990s found AI, as an academic field, arguably in the doldrums. While steady, if slow, progress in algorithms continued, the main story was on the hardware side. The relentless advance of Moore's Law brought into the realm of feasibility—using fairly prosaic algorithms—some problems in AI that had previously been thought to require conceptual or "cognitive" breakthroughs. Poster-child for this trend was the 1997 decisive victory of IBM's Deep Blue computer over world chess champion Garry Kasparov,[13] which was enabled by a modest amount of machine chess expertise combined with a (then) spectacular amount of blind, high-speed searching ability.

Chess, once thought to be a game of subtle human strategy, fell not to a machine with cognitive abilities that mimicked human thought, but to one with a very special-purpose, and very fast,

---

[10] https://en.wikipedia.org/wiki/Dartmouth_Conferences

[11] Marvin Minsky and Seymour Papert, 1969, *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, Cambridge MA, ISBN 0-262-63022-2.

[12] https://en.wikipedia.org/wiki/Expert_system

[13] http://www.nytimes.com/1997/05/12/nyregion/swift-and-slashing-computer-topples-kasparov.html

search algorithm. We emphasize this point, because it is a theme repeated (though not often so dramatically) in the development of AI as a field.

As another example: Shortly after Deep Blue's victory in 1997, a New York Times article about the game of Go asserted that Deep Blue's "roughshod approach [to chess] is powerless against the intricacies of Go." The article went on to say that,

> "…to play a decent game of Go, a computer must be endowed with the ability to recognize subtle, complex patterns and to draw on the kind of intuitive knowledge that is the hallmark of human intelligence…. When or if a computer defeats a human Go champion, it will be a sign that artificial intelligence is truly beginning to become as good as the real thing."[14]

Yet, as it turned out twenty years later, the surprise 2016 victory of Google's AlphaGo against world Go champion Lee Sedol,[15] did not involve any breakthrough in general machine cognition. Instead, it involved a hybrid of DNN technology—the DL revolution that we next discuss—with (as a generation earlier for Deep Blue but now even more so), massively parallel tree-search and reinforcement learning.

Most applications of artificial intelligence today that touch on image classification, object detection, speech recognition, and natural language understanding use deep neural networks (DNNs). For many problems, a DNN can be trained to generate a desired output in response to an input as long as a large *training set* of input-output pairs exists. The DNN is trained to learn the input-output relationship and then can *generalize* its learning to inputs that it has not yet seen. For example, given a training set of animal images labeled with the name of the animal, the DNN learns to identify an animal given an image. DNNs have been trained to classify images, detect objects, identify people from faces, generate text from speech, translate natural languages, and many other tasks. For many of these tasks, DNNs have achieved performance that exceeds what humans typically do.

[14] http://www.nytimes.com/1997/07/29/science/to-test-a-powerful-computer-play-an-ancient-game.html
[15] https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

3 THE DEEP LEARNING REVOLUTION

The basic constructs of deep neural networks [16] [17], convolutional neural networks [18], and back-propagation [19] were all in place by the 1980s. However, DNNs did not become the technology of choice for many applications until after 2010. The delay of more than 20 years in the application of DNNs to key problems was because two key ingredients were missing: labeled data sets for training, and sufficiently powerful hardware for training. For image classification, large training data sets (e.g., ImageNet) were available by 2005, but it wasn't until Alexnet, trained on GPUs, won the 2012 ImageNet competition, that the use of DNNs for image processing became widespread. With the availability of powerful GPUs it became feasible to train large networks on the large data sets needed to achieve good accuracy. Today, the size of a network and the size of the data set on which it is trained remains limited by available hardware for training.

The impact of Deep Learning on multiple sub-fields of AI in just the last five years is nothing short of revolutionary. Using DNNs, between 2011 and 2015, the error rate for image captioning by computer fell from 25% to about 3%, better than the accepted figure for human performance of about 5%.[20] Figure 1, from work done at Google, shows examples of computer captioning. DNNs have exceeded human performance on many tasks, including face recognition, object detection, and speech understanding. (According to Google's speech group, its single word recognition error rate fell, in two years between 2013 and 2015, from 23% to 8%.) Due in large part to the DL revolution, we are in the surprising position that the rollout of self-driving vehicles is now more limited by the speed of policy change than by its technical readiness.

---

[16] Aleksei Grigorevich Ivakhnenko and Valentin Grigorevich Lapa, *Cybernetic predicting devices*, CCM Information Corporation, 1965.
[17] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jakel, *Backpropagation applied to handwritten zip code recognition*, Neural Computation **1**(4), 541-551 (1989).
[18] K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics **36**(4), 193-202 (1980).
[19] D.E. Rumelhart, G.E. Hinton, R.J. Williams, *Learning representations by back-propagating errors*, Cognitive Modeling **5**(3), 1 (1988).
[20] http://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

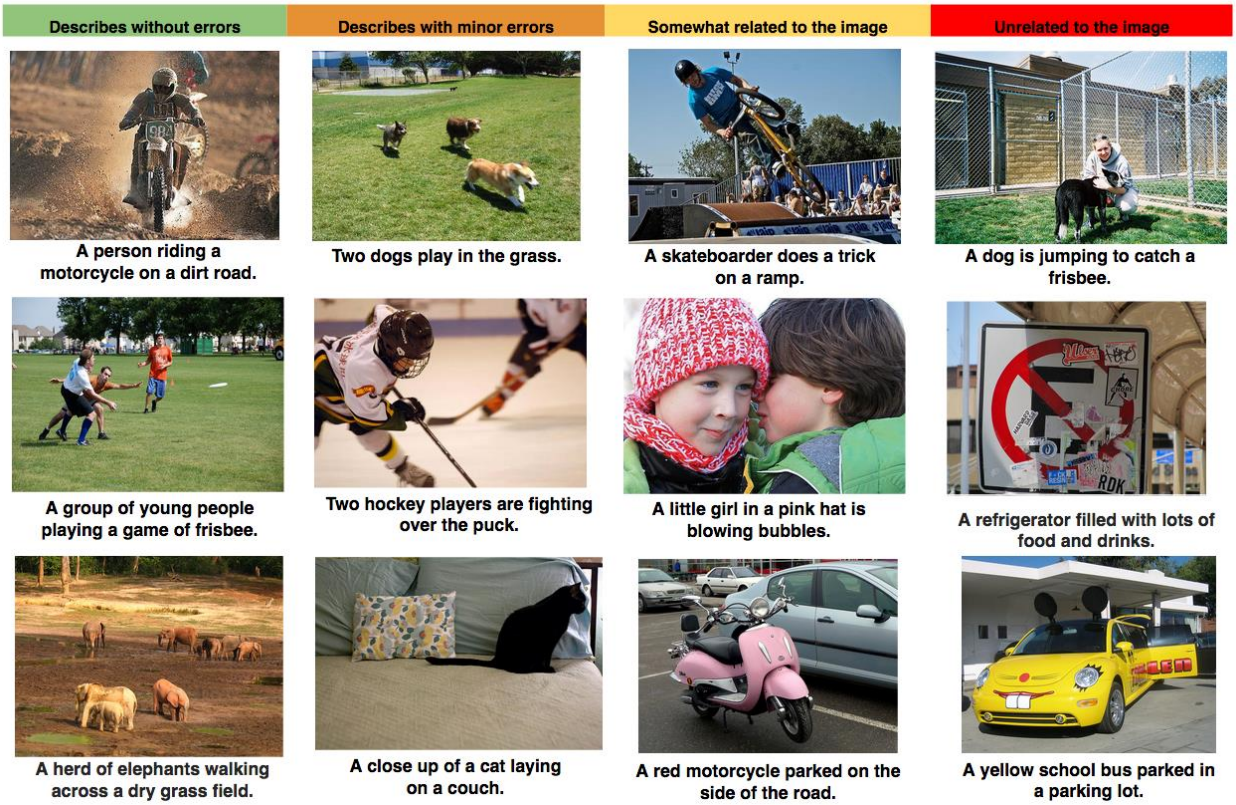

Figure 1. At the current state-of-art, more than 95% of images can be captioned as well as shown in the first column, with the remaining 5% distributed across the other three columns. Note that the errors made are often unintuitive to a human observer (as in the upper right image). Source: Vinyals et al., "Show and Tell: A Neural Image Caption Generator" (2015).

Figure 2 shows examples of computer vision capabilities beyond simple object recognition. There is no clear, bright line between these capabilities and those, not yet achieved, that we may think still require generalized human intelligence (that is, AGI and not just AI). For example, in the context of Figure 2, it is beyond the capabilities of today's machine systems to answer (or discuss) the question, "Will person B put some money into Person C's tip bag?" However, as we will discuss below, today's remarkable capabilities are enabled by the existence of large, relevant data sets. It is by no means impossible that a future data set capturing a sufficiently large range of human actions and activities would enable DL to plausibly discuss the "tip bag" question.

In the remainder of this chapter, we take a pedagogical "technology deep dive" into Deep Learning (DL), deep and convolutional neural networks (DNNs and CNNs), and related matters.[21]. The magnitude of the importance of these technologies is hard to overstate, and we thus feel that technically minded, non-expert readers should be exposed to at least some of the details "behind the curtain". Others may skip to Section 3.4.

[21] Our discussion draws heavily on the book *Neural Networks and Deep Learning* (neuralnetworksanddeeplearning.com) by Michael Nielsen, and the book *Deep Learning* (www.deeplearningbook.com) by Ian Goodfellow, Yoshua Bengio and Aaron Courville.

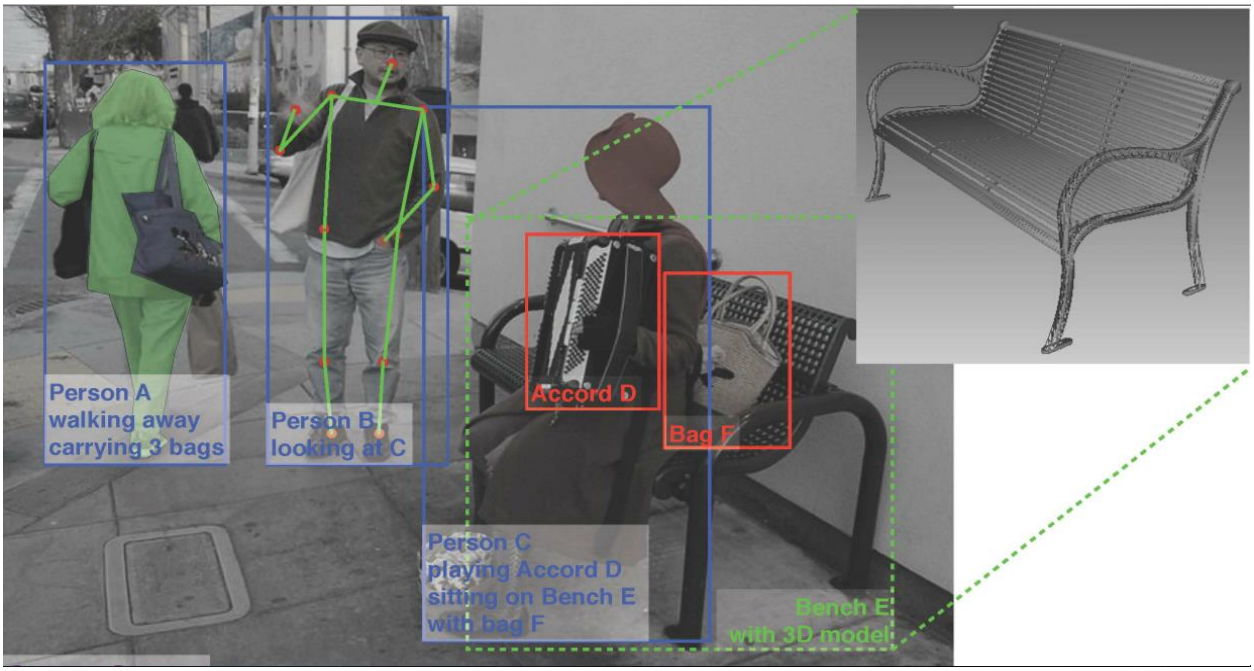

Figure 2. Beyond captioning and object identification, computer vision using DNNs is capable of (a) image segmentation—identifying the pixels that constitute Person A, (b) pose estimation—using a knowledge of human anatomy to construct a likely three-dimensional pose of Person B, (c) associating groups of objects, as that Person C is playing an accordion, (d) recognizing and constructing 3-D models of partially hidden objects, as here the bench. Source: Jitendra Malik.

3.1 Perceptrons

The earliest neural nets, of the 1950s and 1960s, were composed of perceptrons [22]. The perceptron was motivated by a biophysical model of the neuron, and thus the building block is now referred to as a neuron. Each neuron had several inputs, $x_1, x_2, x_3, \ldots\ x_n$, and a single output referred to as its activation, a. The perceptron performed a linear sum over the input to obtain the activation, $a = \sum_{i=1}^{n} w_i x_i + b$, where the weights w_i and bias b define the function of the neuron. This neuron is completely linear. Adding larger layers of neurons increases the dimensionality of the data representation, but the fundamental process remains that of a linear filter. Thus it can be reduced (by matrix multiplication) to a single linear transformation, showing that the intermediate layers are, in fact, illusory. Nevertheless, these early learning networks, as linear discriminators, could do classification on some model problems. The process of discovering the weights and biases such that the network of neurons performs the proper function is called "training the network", and will be discussed in detail in a later section.

[22] Frank Rosenblatt, The Perceptron - a perceiving and recognizing automaton, Cornell Aeronautical Laboratory, Report 85-460-1, (1957)

3.2 Generic Network Architecture

As shown in Figure 3, a *neural network* is a weighted, directed graph where the vertices represent neurons and the edges represent connections (sometimes called synapses) between the neurons. The neurons are arranged in layers. In this simple example the network consists of an input layer with 8 neurons, three hidden layers each with 9 neurons, and an output layer consisting of four neurons. These are shown as fully connected layers, as the output of each neuron is connected to the input of every neuron in the following layer.

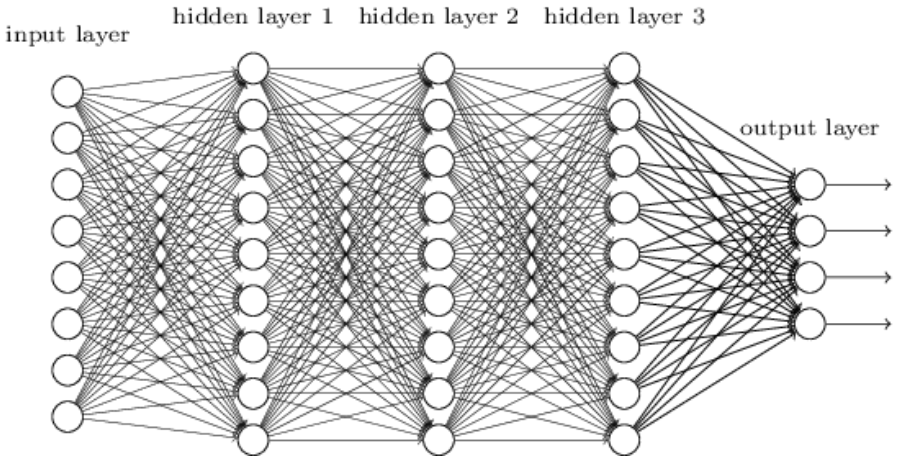

Figure 3. A simple deep neural network (DNN). This network consists of five layers: an input layer with eight neurons, three hidden layers with nine neurons each, and an output layer with four neurons. Layers in modern neural networks have thousands to millions of neurons in dozens to hundreds of layers. Source: neuralnetworksanddeeplearning.com/chapt6.html.

The input stimulus for the network is applied to the input layer. For a network that operates on images, each input neuron represents a pixel (or one color channel of a pixel). An input neuron may represent an audio sample for a speech recognition network or a character for a natural language understanding network. Discrete data, like characters or numbers, are represented in a *one-hot* representation where a separate neuron is used for each possible symbol in each position. For example, a string of *n* letters would be represented with 26*n* neurons one neuron for each possible symbol in each position.

The intermediate neurons represent features that are detected in the image, audio stream, or other input. It is often the case that the hidden layers are of higher dimensionality than the input layers, as it is often useful to expand the problem into a higher dimensional manifold.

The right-most set of neurons represents the output of the network. For example a network that classifies images into one of 1,000 categories (as in the ImageNet competition) would have 1,000 output neurons, one for each category. Output neuron activations are typically computed using a *soft-max layer*, which regularizes the output to that of a probability.

Modern neural nets typically involve many thousands to millions of neurons in a single layer, and deep neural networks involve many tens to hundreds of layers. The result is often billions of weights and biases which need to be calculated.

3.3 Neuron Non-Linearity

The big, early advance in neural networks came with the realization that adding nonlinearity to the neuron dramatically increases the functionality of the system, making it much more powerful that just a simple linear discriminant. The non-linearity is obtained by defining the sum over weights and bias for an individual neuron as an internal variable $z = \sum_{i=1}^{n} w_i x_i + b$, and then applying a non-linear function to z so as to obtain the output[23]. An example of this non-linearity used in early NNs is the sigmoid a=1/((1+exp(-z))), shown in the left panel of Figure 4. The function is linear near to z=0, but saturates for large values of |z|. Its derivative is shown in the right panel of Figure 4. The derivative will play an important role in the training of coefficients. Note that the maximum value of the derivative is 0.25, and that the derivative goes to zero for large values of $|z|$. While it is the non-linearity that increases the usefulness of the neurons, the saturation for large values of $|z|$, and subsequent small value of its derivative, makes the networks challenging to train.

Modern DNNs more commonly use the rectified linear unit (ReLU), $y = \max(0, z)$, as the source of non-linearity [24]. It generally outperforms the sigmoid, as it doesn't saturate for large values of z. The resulting network is easier to train and remains responsive for large values of z.

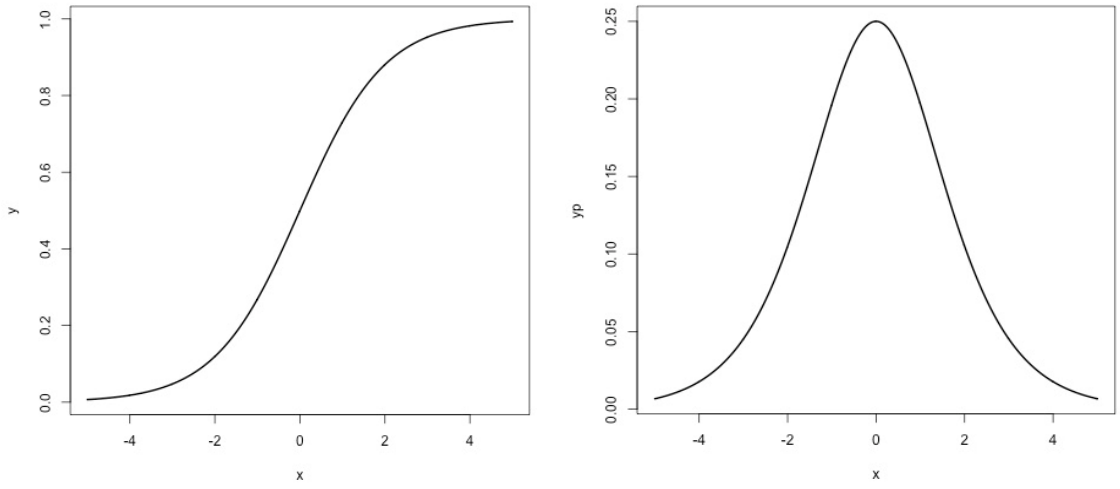

Figure 4. Left, the non-linear sigmoid function, $1/(1 + exp(-z))$, and right, its derivative. Note that its maximum value of the derivative is 0.25, and that it goes to zero for large value of $|z|$.

The introduction of non-linear transformations at the level of the neuron enables the network to implement non-linear transformations of very high dimensional manifolds. This allows classification algorithms to disentangle the what would otherwise be an unresolvable classification problem. As a simple example of how this works, consider the problem of separating the entangled red and blue spiral curves of Figure 5. A simple linear discriminant cannot solve this problem, as there is no straight line that separates the curves. However, after

[23] Note that the biases *b* will be viewed as weight w_0 associated with the affine, $x_0 = 1$.
[24] V.Nair and G.E. Hinton, *Rectified linear units improve restricted boltzman macnines*, ICML, 2010.

transformation by a network containing four hidden layers, the manifold (shown at the top of the page) is deformed such that the spirals are easily separated by a line.

Additionally, it often helps to embed the problem in a higher dimensional space. Shown schematically in Figure 6 is the problem of trying to separate the circular targets from the white background. In the two dimensional manifold there is no straight line which solves the problem. However, if the problem is embedded in 3-dimensions, as shown on the right, it becomes possible to deform the manifold such that a single plane solves the problem. Thus, the higher dimensional space allows a simpler solution of the problem. For this sort of reason, the hidden layers often have considerably more neurons than the input layer.

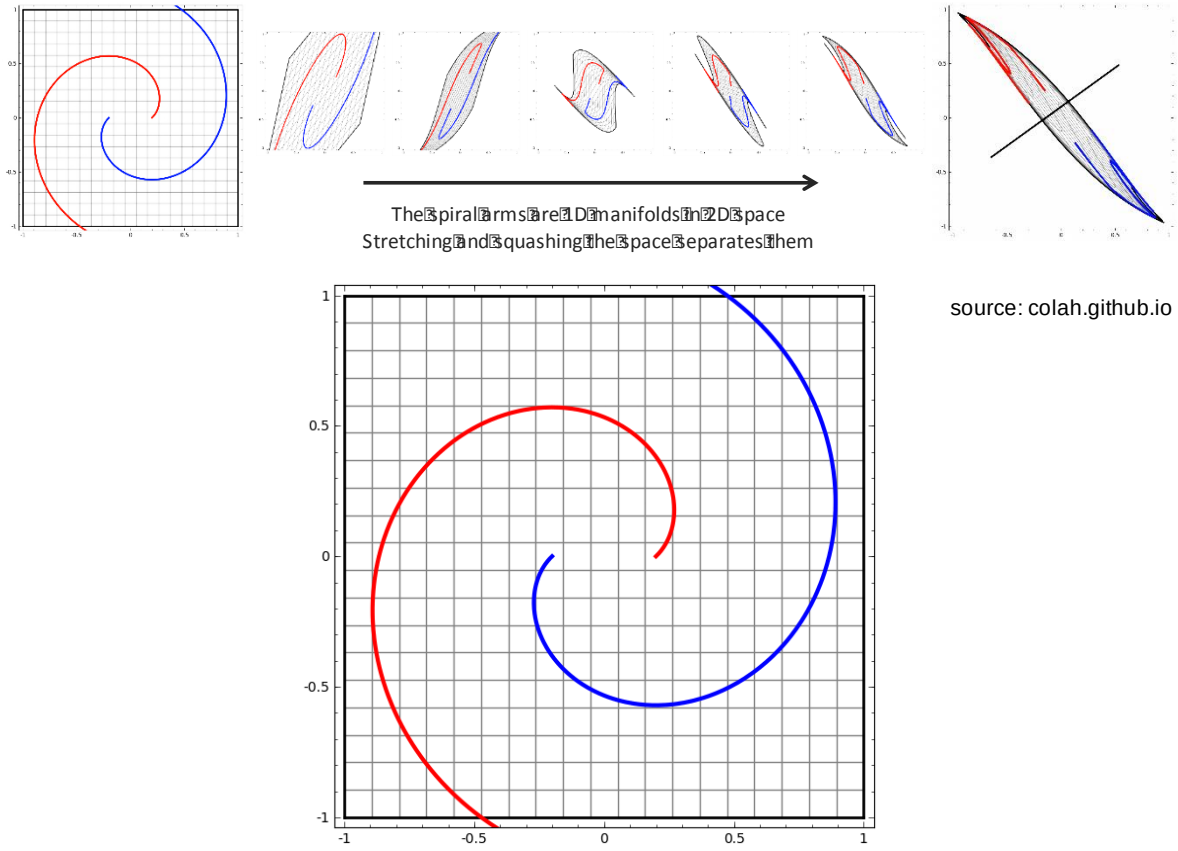

Figure 5. An example of the power of non-linear transformation for classification. The red and blue spirals are entangled such that no straight line (i.e. linear discriminator) can separate them. However, after processing by a network containing four hidden layers, the manifold (shown at the top of the page) is transformed such that the spirals are easily separated by a line. Source: colah.github.io.

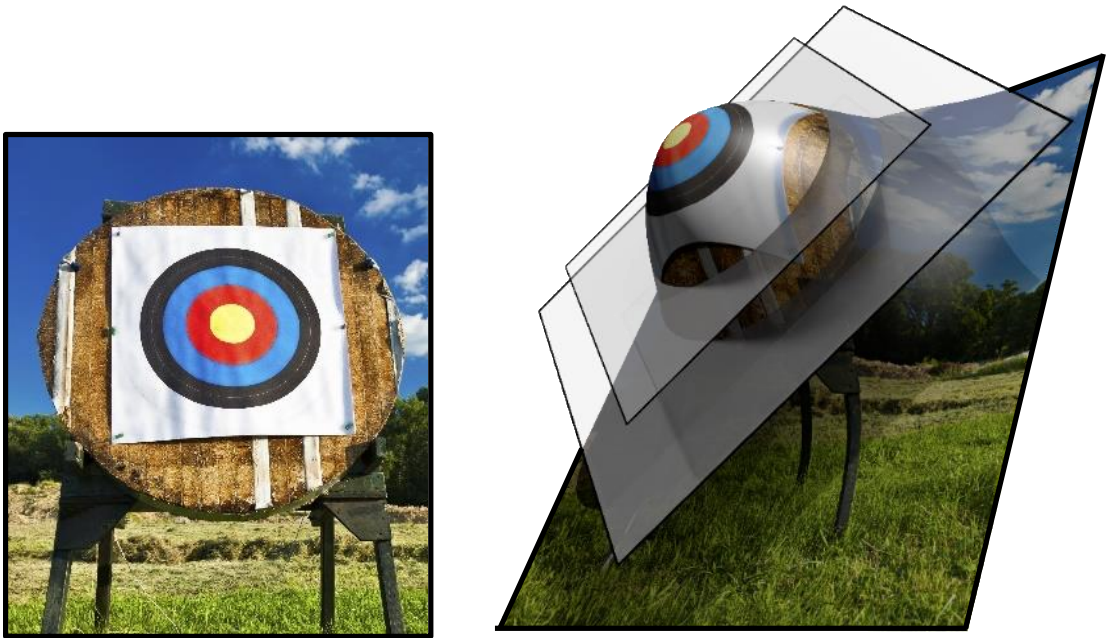

Figure 6. Consider the problem of trying to separate the circular targets from the white background. In this two dimensional manifold, there is no straight line which solves the problem. However, if the problem is embedded in 3-dimensions, as shown on the right, it becomes possible to deform the manifold such that a single plane solves the problem. Source: DARPA brief.

3.4 Training and Back-propagation

Central to the DNN is its training, in which the millions of weights which connect the various neurons are assigned values. The modern technique for training these networks has its origins in a 1986 paper by D. Rumelhart, et al. [25] Understanding this back-propagation algorithm is central to understanding why DNNs have benefited so greatly from big data and GPUs. It also helps to illuminate some of the challenges associated with designing DNNs.

Appendix C has a simple derivation of the back-propagation algorithm, in the spirit of what a non-expert would need to know to understand the major features. This section summarizes the the results derived in Appendix C.

Assume a network with L layers of neurons. The weights and biases in the network are initialized using a random number generator. To train the network, one has N coupled input/output pairs (x,y), where x is a vector spanning the space of all input neurons, and y spans the space of all output neurons.

The general training algorithm then proceeds as follows. For each input/output pair of training data (x,y)

1) Set the activations of the input layer a^1 to the values of the input vector x.

[25] D.E. Rumelhart, G.E. Hinton, R.J. Williams, *Learning representations by back-propagating errors*, Nature **323**(9), 533-536 (1986).

2) Starting with the first layer, forward propagate the input weights z^l and activations a^l layer by layer through the entire network, where

$$[z^l] = [w^l]\,[a^{l-1}]$$
$$[a^l] = [f(z^l)]$$

Note that $f(z)$ is the non-linear function each neuron applies to its input weight *z*.

3) Use the final layer activations a^L and the calibrated output *y* to calculate the error function. If using a quadratic error function, it is of the generic form

$$E = \frac{1}{2}[y - a^L]^T\,[y - a^L]$$

4) Back-propagate the error through the network, obtaining corrections to the weights which minimize the error function for that particular (x, y) pair. Starting with the final layer *L*, the correction to the final layer weights $[\delta w^L]$ are obtained as

$$[\epsilon^L] = [f'(z^L)]\,[a^L - y]$$
$$[\delta w^L] = -\eta\,[\epsilon^L]\,[a^{L-1}]^T$$

The correction to weights in the remaining layers, $[\delta w^l]$, are calculated using the following relations.

$$[\epsilon^l] = [f'(z^l)]\,[w^{l+1}]^T\,[\epsilon^{l+1}]$$
$$[\delta w^l] = -\eta\,[\epsilon^l]\,[a^{l-1}]^T$$

The parameter η is the learning rate. This is back-propagation because the weighted gradients, ϵ^{l+1}, from layer $l+1$ are used to calculate the results in layer *l*. Note that all the values for z^l and a^l are calculated in the forward propagation.

5) Repeat this algorithm for all input/output pairs in the training data. The final corrections to the weights are obtained by averaging over the results obtained for all input/output pairs.
6) Update the weights.

Each iteration through the entire set of training data is an epoch. Many epochs are calculated, until the error function converges to a minimum.

That the entire back-propagation algorithm can be written in a matrix algebra formalism renders it perfectly suited to benefit from the parallelism of graphics processor units (GPUs). Graphics processors were initially developed for processing image data for displays, which are also dominated by matrix operations. The benefits become all the more important as the scale of the problem grows, both in terms of the number of neurons per layer and the total number of layers. Thus, the success of deep neural networks relies on availability of GPU hardware.

3.5 Convolutional Neural Nets

While images can be processed by a network composed of fully-connected layers, like that of Figure 3, they are more efficiently handled by a network that employs convolutional layers. As will be seen below, convolutional neural networks (CNNs) use the translational invariance of image features to dramatically reduce the number of weights in the network.

Consider the input layer and first hidden layer of neurons shown in Figure 7 and Figure *8*. Instead of building a fully connected link to the next layer, a convolution neural network will take a small, spatially related, group of input neurons and connect them to a neuron in the next hidden layer. In the case of this image, the network takes a 5x5 neuron region in the upper left of the input layer and connects each of these neurons to a single neuron in the next hidden layer. The 5x5 region in the input layer is called the "local receptive field". The local receptive field is then shifted over by some distance (called the stride length, here 1 neuron), and this shifted local receptive field is connected to the next neuron in the following hidden layer. This is continued until the entire hidden layer is filled.

The weights defining the connectivity between the neurons in the local receptive field of the input layer and the corresponding neuron in the hidden layer are determined using the same training exercises. However, every neuron in the hidden layer uses the same set of weights. Thus, the z-value for the $(i,j)^{\text{th}}$ neuron in the hidden layer is

$$z_{i,j} = b + \sum_{k=0}^{4} \sum_{l=0}^{4} w_{k,l}\, a_{i+k,j+l}$$

The values $w_{k,l}$ are the same for the entire hidden layer. Thus, in the above example, there only 26 coefficients for the entire hidden layer (i.e. 25 weights plus a bias). For a three color image, there would be 76 coefficients (i.e. 1 + 25*3).

This convolutional architecture works because an image feature is translationally invariant. By using the same set of weights and bias for each neuron in the hidden layer, each hidden layer neuron measures whether or not a particular feature exists in the particular local receptive field of the previous layer to which the neuron is connected. The shared weights define a kernel or filter.

Suppose the l^{th} layer of a CNN is meant to detect multiple distinct features in the l-1^{th} layer. The l^{th} layer is designed as a stack of what might otherwise be viewed as some number of individual layers, each detecting a distinct feature. Thus, the l^{th} layer has the structure (i, j, f). The planes of constant f in each layer are referred to as feature channels. As an example, the input layer typically has only three feature channels (for the red, green, and blue color channels) or only a single channel for a monochrome image.

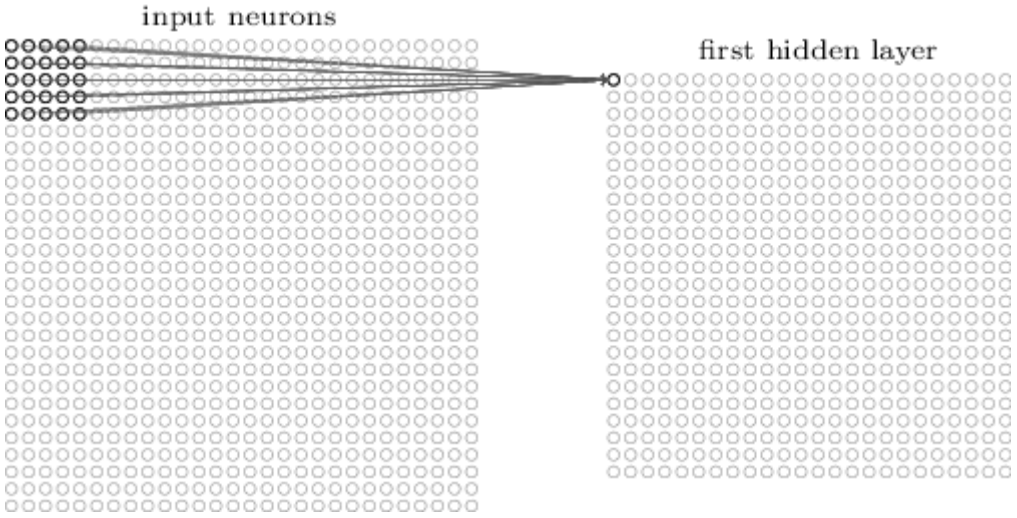

Figure 7. A schematic representing the input and first hidden layer in a convolutional neural network. This network takes a 5x5 array of neurons in the input layer and links them to a single neuron in the first hidden layer. Source: neuralnetworksanddeeplearning.com/chapt6.html.

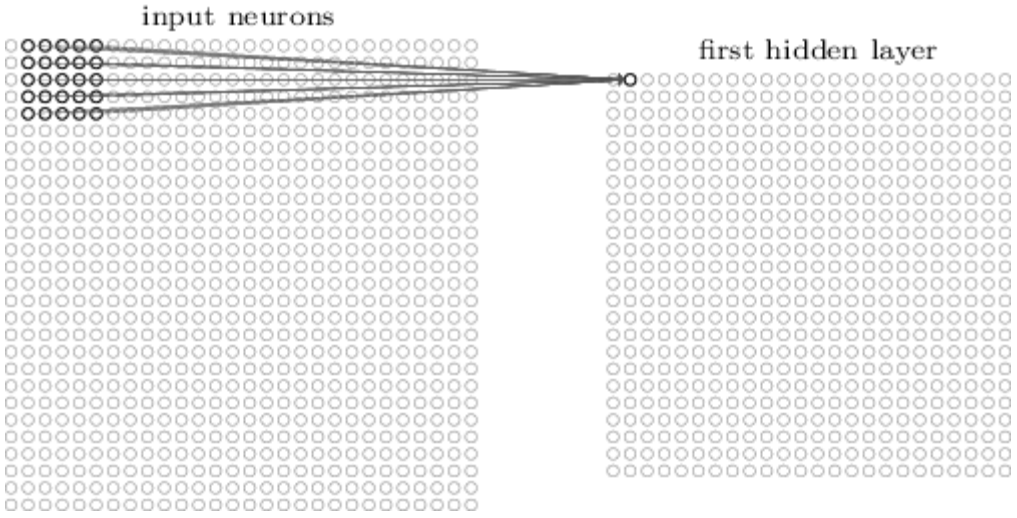

Figure 8. The second neuron in the first hidden layer is connected to a 5x5 array of neurons in the first input layer, shifted by one column relative to what was shown in Figure 7. Source: neuralnetworksanddeeplearning.com/chapt6.html.

The next step in modern convolutional nets involves a pooling layer. A pooling layer condenses the feature map by taking small, non-overlapping regions of neurons in the feature map and pooling them together. Pooling layers are typically 2x2 neuron filters with a stride of 2. A max-pooling layer takes the maximum value of the activations of the neurons to be pooled. There are various types of pooling that could be done (mean pooling, L2-norm pooling, etc.). Pooling reduces the size of the representation, and thus reduces over-fitting to the training data.

In summary, Figure 9 shows a schematic of what might be a typical convolutional layer in a network. The input is a 28x28 pixel image. The image is processed with a 5x5 convolutional kernel and a stride length of one pixel, yielding 24x24 neuron feature maps. Here there are are three feature maps, each with its unique set of weights. Each feature map is pooled with a 2x2 pooling layer, yielding three 12x12 feature maps.

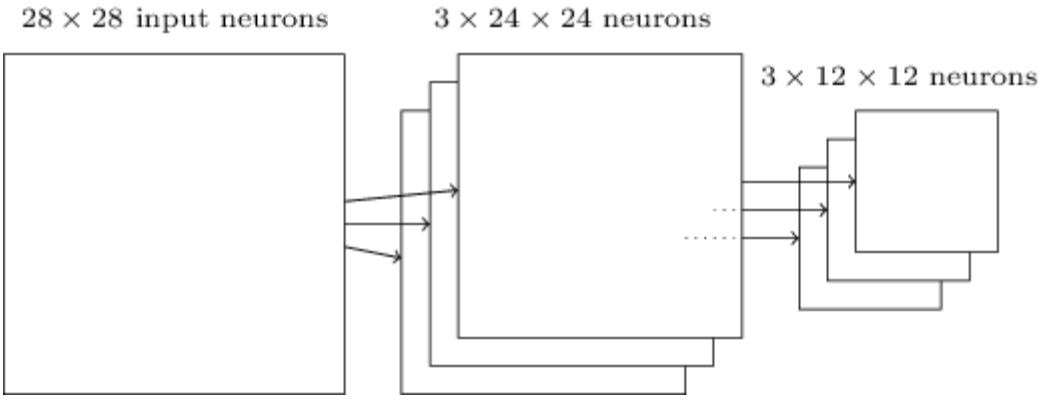

Figure 9. A schematic of a typical layer in a convolutional network. The input is a 28x28 pixel image. The image is processed with a 5x5 convolutional kernel and a stride length of one pixel, yielding 24x24 neuron feature maps. Here there are are three feature maps, each with its unique set of weights. Each feature map is pooled with a 2x2 pooling layer, yielding three 12x12 feature maps. Source: neuralnetworksanddeeplearning.com/chapt6.html .

3.6 Some Details of DNNs

The following is an assortment of topics relevant to understanding DNN form and function.

3.6.1 Error functions

Error functions are also called either cost functions or loss functions in the DNN literature. Many different types of error functions have been used in DNNs. Quadratic error functions are perhaps the simplest. Others include cross-entropy functions, which when used with the sigmoid non-linearity reduce the lack of sensitivity of the back-propagation algorithm at large values of $|z|$.

3.6.2 Softmax layer

The activations of the final neural layer of a DNN is often transformed by a soft-max layer. Given a set of activations *i* in the final neural layer *L*, a_i^L, the output of the soft-max layer is

$$y_i = \frac{\exp(a_i^L)}{\sum_i \exp(a_i^L)}$$

The point to the layer is to transform the activations a_i^L into a set of values which have the properties of probabilities. In particular, $y_i > 0$ and $\sum_i y_i = 1$.

3.6.3 Training, validation, and test data

The complete set of input/output pairs used to train the network is typically randomly divided into three groups: a training set, a validation set, and a test set. Training data are used to train the network, as was described above in the section on back-propagation.

Validation data are used to measure how well the training is progressing and to adjust various parameters used in the training, such as the learning rate, the network architecture, etc. Additionally, you might evaluate the DNN against the validation data after each epoch as a benchmark for progress.

When the training process is completed, the test data are used to measure the success of the final DNN.

3.6.4 Stochastic gradient descent

The back-propagation algorithm is based on gradient descent. Stochastic Gradient Descent makes use of the principle that one can make a reasonable estimate of the gradient by sampling over a small, randomly selected, set of input/output pairs. Using these mini-batches helps to speed up the learning process. For example, suppose you have *N* sets of training data, and these are randomly divided into *M* mini-batches of size *N*/*M*, such that each training datum occurs once. For big data, the typical size of *N* is of order 10^6, and the typical size of *M* is of order 10s of (x, y) pairs. One iteration of the back-propagation algorithm is run for each mini-batch, and thus the weights are updated for each mini-batch. Doing this for all *M* mini-batches corresponds to one epoch. After the first epoch is finished, the data are again randomly divided into mini-batches, and again the weights are updated for each mini-batch. This is then continued through several epochs, typically several 10s of epochs. Thus, stochastic gradient descent is defined in terms of the number of mini-batches and the number of epochs.

There are many variants on stochastic gradient descent. For example, momentum based stochastic gradient descent averages the gradient of the error function over the past several mini-batches.

$$v_i = \mu v_{i-1} - (1-\mu)\eta \frac{\partial E}{\partial w}$$

$$w_i = w_{i-1} + v_i$$

where the parameter *i* here cycles through subsequent mini-batches.

3.6.5 Weight-decay: L2 and L1 regularization

Weight decay is a regularization method that reduces over-fitting by conditioning the network to learn small weights. Keeping the weights small makes the network less sensitive to noise in its inputs, and thus less likely to learn the noise. As an example of weight decay, L2 regularization adds a penalty quadratic in the weights to the error function, $E = E_0 + \frac{\lambda}{2N}\sum_i {w_i}^2$, where $\lambda > 0$ is the strength of the regularization penalty and the sum over *i* is meant to include all weights. As is described below, this sum over weights does not normally include the biases.

Taking the gradient,

$$\frac{\partial E}{\partial w_i} = \frac{\partial E_0}{\partial w_i} + \frac{\lambda}{N} w_i.$$

On training, the weight becomes

$$w_i \to w_i - \eta \frac{\partial E}{\partial w_i} = \left(1 - \frac{\eta\lambda}{N}\right) w_i - \eta \frac{\partial E_0}{\partial w_i}.$$

Thus, it is clear L2 regularization serves to reduce the size of the weights.

Another example of weight decay is L1 regularization, $E = E_0 + \frac{\lambda}{2N}\sum_i |w_i|$. L1 regularization tends to drive small weights to zero faster than L2 regularization, but shrinks large weights slower than L2. Thus, L1 regularization generally results in networks with a sparse set of important connections.

While one could also regularize the biases, in practice it is not normally done. Having a large bias is different than having large weights. Large weights increase the sensitivity of the neuron to its inputs, and thus makes it more likely to over-climb on input noise. In contrast, large biases make it easier for the neuron to go into saturation, which can be a desirable property in that it desensitizes the neuron to all inputs, but isn't likely to result in over-climbing on input noise.

3.6.6 Dropout

Dropout is another technique used in the training of the network that helps to prevent over fitting. For each mini-batch, a different a different set of randomly selected neurons is omitted from the network. Typically, of order half the neurons in the hidden layers of the network are omitted. Training for a mini-batch occurs as normal, except the network architecture excludes the omitted neurons. Thus, dropout implements training over a reduced network. Dropout reduces over-fitting by making it more difficult to climb on correlations between neurons.

3.6.7 Unstable gradients and unbalanced leaning in DNNs

As was discussed in the section on back-propagation, the correction to the weights in layer *l* is given by the expression

$$[\delta w^l] = -\eta\, [\epsilon^l]\, [a^{l-1}]^T$$

where

$$[\epsilon^l] = [f'(z^l)]\, [w^{l+1}]^T\, [\epsilon^{l+1}]$$

This expression for $[\epsilon^l]$ can be written explicitly for all layers through to the final layer *L* as the product,

$$[\epsilon^l] = \left[f'(z^l)\right]\left(\prod_{i=l+1}^{L} \left[w^i\right]^T \left[f'(z^i)\right]\right)[a^L - y]$$

This product is made of terms of the general form $w^i f'(z^i)$. As was shown in Figure 4, the derivative of the sigmoid functions peaks at 0.25, and is generally much smaller than that. If the weights are a normal random variable, than it will generally be the case $\left|w^i f'(z^i)\right| < 1$. Thus

the parameters ϵ^l tend to get smaller as one works towards earlier layers. This is called the vanishing gradient problem. It results in the general rule of thumb that earlier layers tend to learn slower than the later layers.

Of course, it does not have to be the case that $\left|w^i f'\left(z^i\right)\right| < 1$. It is possible to have large weights, $w^i \gg 1$, and biases that center the input weights z^i near to zero where $f'\left(z^i\right) \approx 0.25$, resulting in $\left|w^i f'\left(z^i\right)\right| > 1$. This tends to yield the opposite problem, a diverging gradient, in which earlier layers learn faster than later layers.

The underlying problem is not that the gradient is either vanishing or diverging, rather it is that the gradient is unstable. This comes from the fact that δw^l involves products from terms in all subsequent layers, and the product of many terms is unstable. As a result, different layers will learn at different rates, and the learning is unbalanced. This problem becomes worse as the number of layers become large, and is thus a particular challenge for DNNs.

3.7 Dealing with Small Data Sets

Perhaps the biggest obstacle to adopting the use of Deep Learning in a particular application is the lack of large, labeled data sets for training. In this section we examine some techniques for coping with the lack of labeled data.

3.7.1 Transfer learning

A common technique to construct a network for a small data set without overfitting is to use *transfer learning* in which feature detectors trained on a large data set are used to build a network for a smaller data set.[26] [27] A network, for example VGG16 (discussed in Section 6.1), is first trained on a readily available data set, for example ImageNet. This is done to train the feature detectors in the convolutional layers. The convolutional layers are then frozen and the fully-connected layers are retrained using a small data set. The result is a network that gives much better performance than if it were trained solely on the small data set.

Transfer learning works because, to first approximation, feature detectors are domain independent. Thus, feature detectors trained on one data set work reasonably well on a different data set. With a much smaller set of parameters in the fully-connected layers to be trained, a smaller data set is adequate to get reasonable results.

Transfer learning using a network pre-trained on ImageNet has become a standard protocol for computer vision applications with limited training data.

3.7.2 Data augmentation

A training set of a given size can be made to appear larger through *data augmentation*. Each image in the training set is transformed to create multiple similar images. Training with the

---

[26] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2010.

[27] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1717–1724, 2014.

augmented data set has been shown to lead to better accuracy. Transformations for data augmentation include:

Cropping: a single image may be cropped in multiple ways, moving the central object in the image.

Mirroring: the image is flipped about the vertical axis, swapping left and right.

Warping: the image is warped, for example by "keystoning". More complex, non-affine, warps may also be used.

Exposure and Contrast: The exposure and contrast of the image may be varied.

Color Balance: The color temperature, saturation, red/green balance, etc., of the image are adjusted.

The different transformations can be combined giving a multiplicative number of augmented images. For example, if one combines 5 crops with 2 mirrorings, 5 warps, 5 exposure adjustments, and 5 color adjustments each image expands to 1250 images in the augmented data set.

3.7.3 Autoencoders

An autoencoder is a DNN whose output and input are the same. Thus, for an autoencoder, any dataset is a labeled data set. Autoencoders are typically constructed in a symmetrical manner as illustrated in Figure 10. An input image is fed through a number of convolutional layers, each smaller in spatial dimension but larger in number of channels. At the mid-point of the network an input image is encoded in a high-dimensional intermediate representation. The output layers are then a mirror image of the input layers. They *decode* the intermediate representation to generate an approximation of the input image.

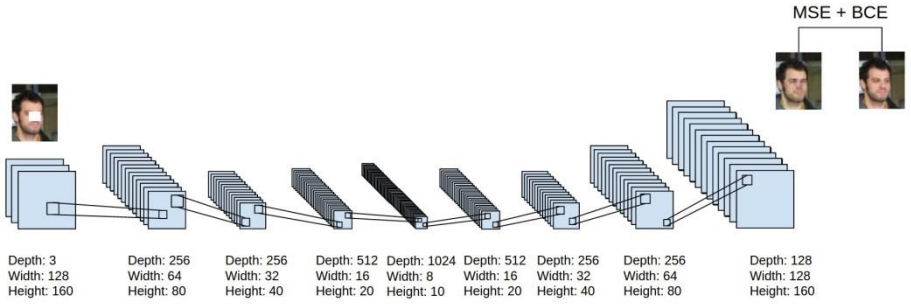

Figure 10. An autoencoder is a DNN that has identical input and output. It is trained by applying an input to the network and then comparing the output to the input image to compute the loss function. Source: Avery Allen and Wenchen Li[28]

[28] http://www.cc.gatech.edu/~hays/7476/projects/Avery_Wenchen/

Autoencoders are useful in their own right for image denoising and upscaling. However, they are also useful to train input layers on unlabeled data sets[29] [30]. One trains the autoencoder on the unlabeled data. Then the output stages are discarded and replaced by a classifier network of one or two fully-connected layers. This classifier network is then trained on a labeled subset of the larger data set while holding the convolutional layers constant.

3.7.4 Recurrent neural networks

Recurrent neural networks (RNNs) are used when the input, output, and/or internal state of a network must deal with memory or sequence. RNNs are typically used to deal with speech, text, image captioning, and other language processing tasks.[31] As shown in Figure 11, a RNN is a neural network with feedback. The feedback causes the hidden state *s* to be a function of all previous inputs, not just the current input. In this manner, state *s* has a *memory* that reflects the history or sequence of inputs to the network. We can analyze an RNN by *unfolding* it in time as shown on the right side of Figure 11.

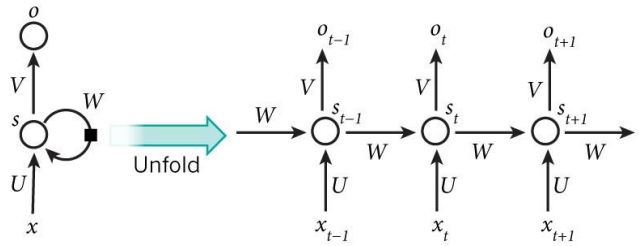

Figure 11. A recurrent neural network (RNN) is a network with feedback that gives it a memory. Hidden state *s* at time *t* reflects the sequence of inputs *x* up until time *t*. An RNN can be analyzed by *unrolling* it in time as shown on the right.

In practice the simple feedback connections in Figure 11 are typically replaced by a more complex memory mechanism. Long- and short-term memory (LSTM) cells[32] or gated recurrent unit (GRU) cells[33] [34] are commonly used for this purpose. A discussion of these cells is beyond the scope of this report.

---

[29] Jonathan Masci, Ueli Meier, Dan Ciresan, and Jurgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[30] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM, 2008.

[31] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 6645–6649. IEEE, 2013.

[32] Martin Sundermeyer, Ralf Schluter, and Hermann Ney. Lstm neural networks for language modeling. In Interspeech, pages 194–197, 2012.

[33] Kyunghyun Cho, et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[34] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR, abs/1502.02367*, 2015.

An RNN can be trained to learn a language model by feeding it sentences one word at a time on the input *i* and presenting the next word of the sentence as the desired output *o*. The words are typically encoded as one-hot vectors (see Section 3.2) over the vocabulary. After seeing several words on *i*, the hidden state of a trained network will be a high-dimensional representation of all of the words seen to that point. This representation can be used, for example, to translate the sentence to a different language or to answer questions about the sentence.

3.8 Summary of the Big Data Deep Learning "Dogma"

The powerful successes of Big Data / Deep Learning have given it the status of a kind of dogma—a set of principles that, when followed, lead often to unexpectedly powerful successes. A brief summary of these principles might be the following:

- Use deep (where possible, very deep) neural nets. Use convolutional nets, even if you don't know why (that is, even if the underlying problem is not translation invariant).
- Adopt flat numerical data representations, where the input is a vector of reals and the internal representation (for a DNN, the activations) is an even larger number of reals. Avoid the use of more complicated data structures. The model will discover any necessary structure in the data from its flat representation.
- Train with big (*really* big) data. Don't load on model assumptions, but rather learn everything from the data—that is where the truth lies. As an example, don't attempt to hardwire the laws of aerodynamics into an autopilot application. With enough data, it is more efficient to let the DNN discover them on its own.
- An approximate answer is usually good enough. When it works, it is not necessary to understand why or how.

4 DEEP LEARNING AND THE "ILITIES"

The recent revolution in Big Data / Deep Learning (BD/DL) will be of importance to many future DoD capabilities; the pressure to incorporate DL into DoD systems will continue to grow. However, as an important caveat, the current cycle of progress in BD/DL has not systematically addressed the engineering "ilities": reliability, maintainability, debug-ability, evolvability, fragility, attackability, and so forth. Further, it is not clear that the existing AI paradigm is immediately amenable to any sort of software engineering validation and verification. This is a serious issue, and is a potential roadblock to DoD's use of these modern AI systems, especially when considering the liability and accountability of using AI in lethal systems.

4.1 An Analogy: Software Engineering as a Discipline

The issue of software engineering and its relation to software development is a subject going back decades. It was codified in a seminal 1990 paper by Mary Shaw[35], where the following question was asked: "Is [in the year 1990] software engineering an engineering discipline?" Shaw distinguished three stages in a field's development, summarized in Figure 12. In the first stage, software development is a craft done by virtuosos and talented amateurs. When it becomes possible for the software to be produced as a commercial product, issues associated with how to produce the product are considered.

However, this doesn't make software development an engineering discipline. For example, the product development can still be done by skilled craftsmen following empirically determined procedures. Software development doesn't reach the stage of professional engineering until there is a science associated with it, including educated professionals, quantitative analysis, etc. Several areas of software development have made the transition to software engineering. Compiler development used to be an art form, but now is a rigorous engineering discipline. Additionally, in many areas of the modern commercial software industry there are well established methodologies for developing and maintaining very large code bases. These methodologies are clearly recognizable (now in 2016) as professional software engineering. They were not so recognizable in 1990.

[35] Mary Shaw, *Prospects for an Engineering Discipline of Software*, IEEE Software **7**(6), 15-24 (1990).

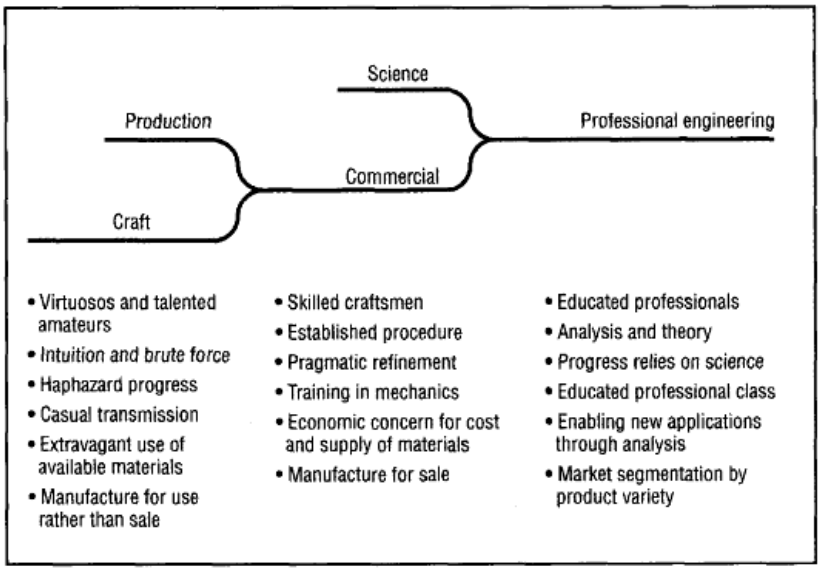

Figure 12. Evolution of an engineering discipline. The lower lines track the technology, and the upper lines show how the entry of production skills and scientific knowledge contribute new capability to the engineering practice. Source: Shaw, footnote [35].

So the question is, by analogy: Where does "AI engineering" fall on Shaw's developmental path, now or in the future?

4.2 Why the Ilities May Be Intrinsically Hard for Deep Learning

One might expect that AI using DL methods will evolve to become an engineering discipline much as software engineering has done. However, the current push for rapid progress in the BD/DL paradigm has moved development away from the rigors of software engineering and its associated "illities". Additionally, a point that we want to emphasize, the very nature of DNNs may make it intrinsically difficult for them to transition into what is typically recognized as a professionally engineered product.

There are two distinct issues for AI engineering. The first is the code on which the DNNs are based. It should be possible to develop this code using principles of software engineering, even in the current environment with the push for rapid progress; but one must commit to the effort. The second issue, the issue that may be much more problematic, is the sheer magnitude, millions or billions of parameters (i.e. weights/biases/etc.), which are learned as part of the training of the net. Their values depend on the particular set of data on which the nets are trained, the order in which these data are processed, the particular training algorithm, etc. Nets optimized for the same job but trained differently will have different parameters, and in the tails will likely give different results.

The final set of parameters define a point in a very large dimensional space, hopefully a point that gives performance close to the global optimal to the particular problem. However, this un-transparent mass of coefficients makes it impossible to really understand exactly how the network does what it does. Thus the response of the network to all possible inputs is

unknowable. This property makes it intrinsically difficult to address the various engineering "ilities" systematically.

As was described above, the data lie on manifolds in very high-dimensional space. For instance, a 1000x1000 pixel image with three colors lives in a 3 million-dimensional space. Related images are not randomly distributed in this space. Instead, they occur on lower dimensional manifolds, as should be clear because one can continuously transform from one image into another. A simple example of this is shown in Figure 13, where pictures of faces of different people in different poses have been collected. A learning system will tease out the coordinates of the manifolds, in this case which can be varied at fixed identity across different poses, or at fixed pose across different identities. This particular example show how low-dimensional intuition can provide a comforting and reassuring view of how DL works.

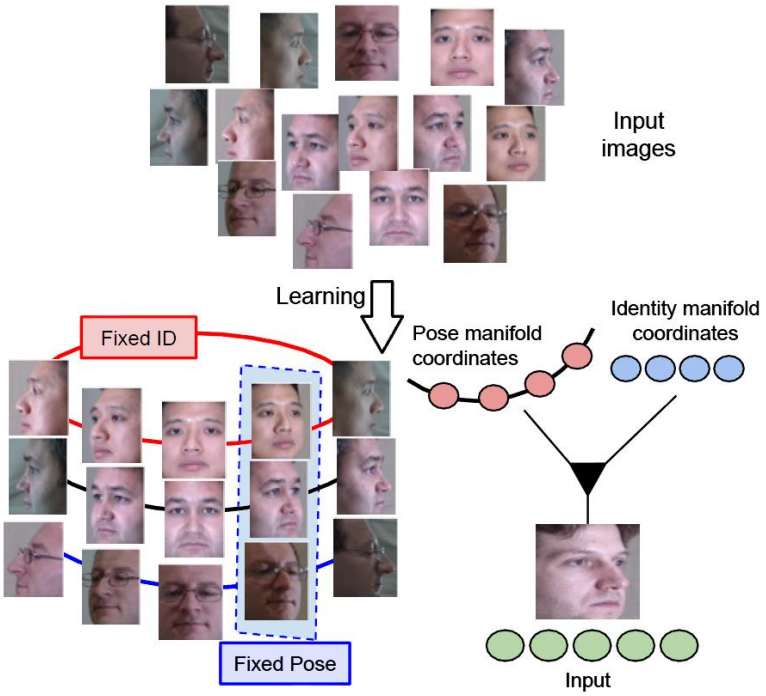

Figure 13. Pictures of faces of different people in different poses. A learning system will tease out the coordinates of the manifolds that can vary at fixed identity across different poses, or at fixed pose across different identities.

Unfortunately, our low-dimensional intuition can give a very misleading view of how DL actually works. An interesting example of this problem[36] is shown in Figure 14, in which the a DNN was given the problem of identifying two animals which have very similar coloring: a gibbon and a panda. The DNN was trained using many labeled pictures of gibbons and pandas, as well as other animals. Our intuition views the classification problem as existing on some low dimensional manifold, shown schematically as being projected into a 2-dimensional manifold in Figure 14. Here, the classification problem is imagined to involve resolving the different domains as being either that of a gibbon or a panda.

[36] I.J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, arXiv:1412.6572v3[stat.ML], 2015.

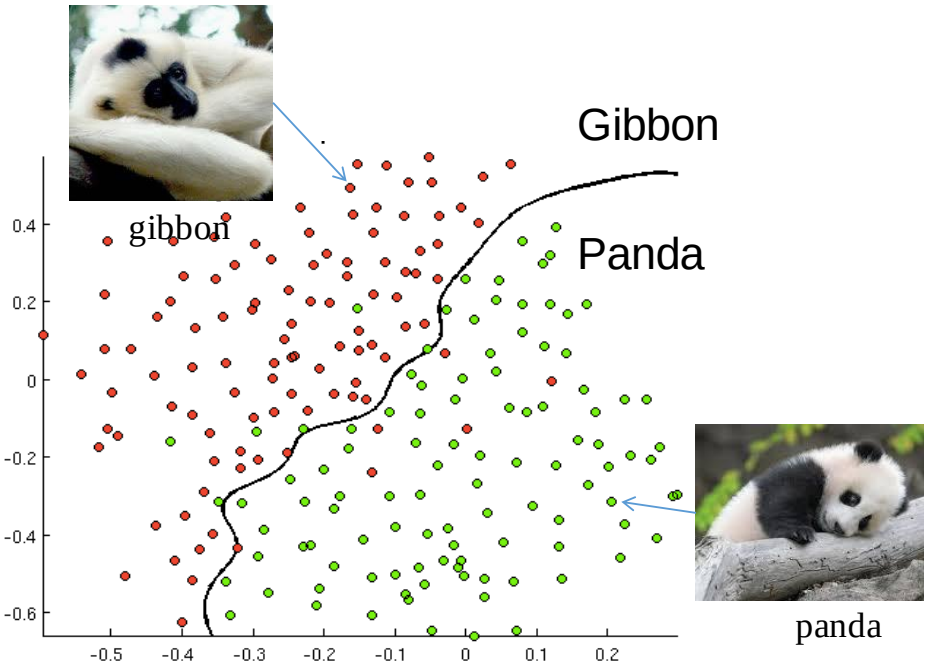

Figure 14. Image of a gibbon and a panda. Our low-dimensional intuition generally imagines some low dimensional manifold, here shown as a two-dimensional space, which allows us to distinguish gibbon from panda. Source: Peter Norvig brief.

The real problem exists, however, in a very high-dimensional manifold, where the boundaries between domains is not always clear, or even correct. These boundaries can be challenged as follows. Consider the image of the panda shown on the left in Figure 15. This image exists at some particular point in the manifold, which the neural network properly identifies as a panda with 57.7% certainty. Starting at this point in the manifold, on can calculate the gradient vector which represents the most direct translation from the panda domain to the gibbon domain. This vector, the center image in Figure 15 looks to us like noise, but the DNN identifies it as the animal "nematode" with 8.2% confidence. Adding this vector to the original picture of the panda with a weight of 0.007 results in the figure on the right in Figure 15. This image still looks very much like the original panda picture, but the DNN identifies it as a gibbon with 99.3% certainty. Thus, the gradient vector has moved the image into a domain of the manifold which the DNN strongly associates with the gibbon. Clearly, this classification is incorrect. It highlights that in a manifold of millions/billions of dimensions, there will always be pockets which are misclassified. While this type of mistake might be tolerable in some commercial systems, it is likely intolerable in DoD systems where the consequences of mistakes are more acute.

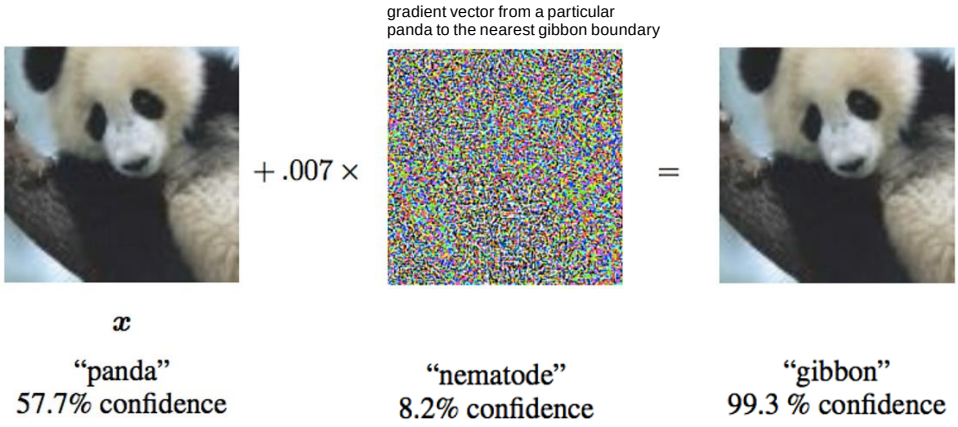

Figure 15: One can get from the panda classification to the gibbon classification by adding what appears to us to be noise. The resulting image looks to us like a panda, but it looks to the DNN like a gibbon, with 99.3% confidence. Source: see footnote [36].

Our low-dimensional intuition misleads us into thinking that the "space of animals" is something with compact regions, as shown on the left side of Figure 16. But of course that can't be right. "Most" images in 1000x1000x3-dimensional image space look like pure noise. Animals comprise tiny lower-dimensional slices. Perhaps like the right-hand panel in Figure 16, then?

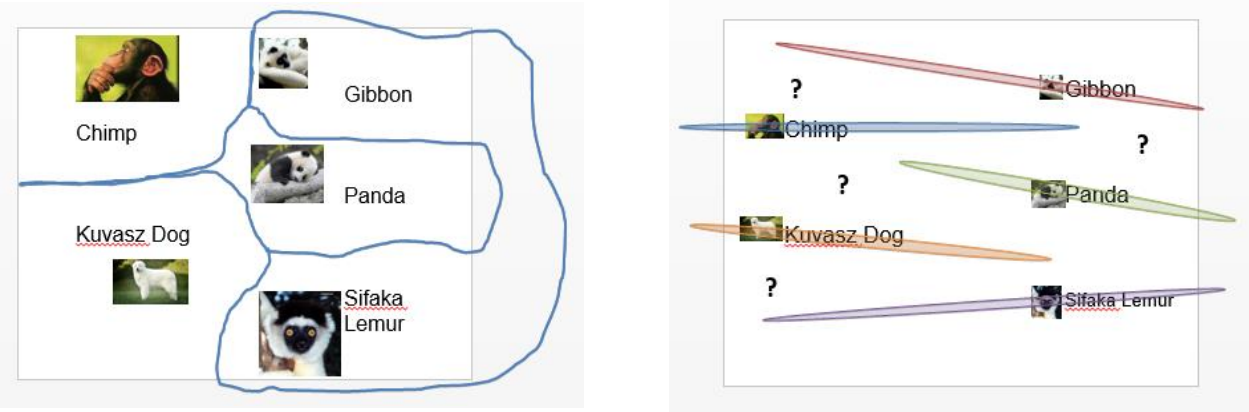

Figure 16. Manifolds of meaningful data are infinitesimal fractions (fractals!) of high-dimension space. Source: Peter Norvig brief.

No such luck! The geometry of data manifolds in high-dimensional spaces is vastly more complicated and impossible to fully characterize than either panel in Figure 16. Paraphrasing J.B.S. Haldane, these manifolds are not just stranger than we imagine, but stranger than we can imagine.[37] As only a slight improvement on Figure 16, we might propose an image like the one shown in Figure 17, an artist's conception. In such a situation, it seems plausible that engineering the "ilities" will be particularly challenging.

[37] The original quote is "Now my own suspicion is that the Universe is not only queerer than we suppose, but queerer than we can suppose." See https://en.wikiquote.org/wiki/J._B._S._Haldane .

Figure 17. A possible artist's conception of the complicated nature of manifolds in high-dimensional space (image © Oleskii Koval)

While the issue of the "ilities" has not impeded the recent rapid development of DNNs, it has not gone unnoticed. The recent paper by D. Scully, et. al. titled *Machine Learning: The High-Interest Credit Card of Technical Debt*[38] laments the lack of understanding of the "ilities" in the latest boom in machine learning. In particular, they suggest the current pace of development is creating a large "technical debt" which will eventually have to be addressed. The term technical debt is meant in the sense that system capability has evolved in performance faster than the corresponding development of deep understanding of what the DNNs are doing, of methods to manage the performance, especially when it goes wrong, and of formal control of the software development and maintenance process. People are creating systems that work, that may become commercial, but which have unknown liabilities downstream.

That the nature of these systems is to be prone to error is one reason why it might be better to treat DNNs as components in a larger, or hybrid, system (see Section 5.4). One can imagine the DNN as one piece of a larger system, where the other pieces have supervisory roles, enabling the robust validation and verification associated with software engineering. Indeed, many of the DL systems which have captured the public's attention are, though for different reasons, hybrid designs, for example Google's AlphaGo system.

[38] D. Scully, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, *Machine Learning: The High-Interest Credit Card of Technical Debt*, Software Engineering for Machine Learning (NIPS 2014 Workshop), 2014

5 AREAS OF RAPID PROGRESS OTHER THAN DEEP LEARNING

While the "Big Data / Deep Learning dogma", as summarized above in Section 3.4, has rightly captured the imagination of experts and the lay public alike, there is some danger of its overshadowing some other areas of AI that are advancing rapidly and hold significant future promise, including in DoD applications. In this Chapter, we review what we think are the most important of these.

5.1 Reinforcement Learning

The basic idea of reinforcement learning (RL) is to let the computer generate its own training data by giving it a metric or score that it should seek to optimize, and then letting it loose in a (generally simulated) environment with a set of possible actions that it can take. Initially the computer's actions will be ineffectual, but gradually, over time, it will learn characteristic states of the environment in which certain actions affect positively the score. DNNs are often used within RL as fast, approximate calculators of the score that is being optimized.

RL (with or without the use of large DNNs as "subroutines") allows training without the necessity of huge labeled data sets. That is, labeled pairs of correct input and output never need to be presented to the machine.

A recent example of successful RL is the Google DeepMind group's systematic learning on 43 Atari 2600 games from the 1970s, including favorites such as Pong, Space Invaders, Breakout, etc. The computer sees only the pixels on the screen, and is told to maximize the score (e.g., as it is displayed in a certain region of pixels) using the set of actions provided by the game joystick. The particular RL algorithm used by DeepMind achieves better-than-human performance for about half of the Atari games. For some games, the performance is vastly superior to humans.[39]

Successful examples that mix RL with labeled data are known. AlphaGo used two DNNs. One was trained on a library of recorded games played by humans (in effect, a labeled data set), and the other was used for estimating the value or score function for RL.

5.2 Graphical Models

A graphical model is a probabilistic representation of a causally related set of random variables (nodes of a graph) that are linked by conditional dependencies. The dependencies may be as general as a complete multivariate joint distribution over the input variables, or any kind of simpler model (e.g., depend only on the sum of inputs). Figure 18 shows a simple example with four variables. At scale, state-of-the-art graphical models may have thousands or more variables.

[39] https://storage.googleapis.com/deepmind-data/assets/papers/DeepMindNature14236Paper.pdf

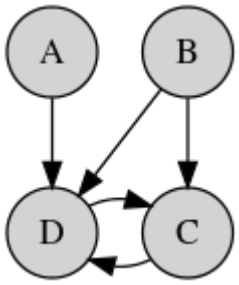

Figure 18. A graphical model with four random variables. Here A and B are independent variables, D depends on A, B, and C; and C depends on B and D. (Source: Wikipedia)

Graphical models are also called probabilistic graphical models (PGMs—not to be confused with precision guided munitions!), and come in various flavors: Bayesian networks (also called belief networks), Markov random fields, conditional random fields, restricted Boltzmann machines, and so on. It is beyond our scope here to explain the differences.

What makes graphical models useful is that they can *assimilate data,* and that they can be *solved* (usually awkwardly termed, "have inference performed on them"). That is, certain of the random variables can be set to measured or known values, after which the probability distributions (including, if desired, joint distributions) of all the other "hidden" variables can be calculated.

The "secret sauce" for a large graphical models lies in the details of the highly developed inference algorithms, and also in finding useful approximations for very large systems. For example, large graphs can be simplified by marginalizing (in the statistical sense) over whole subgraphs. A robot may not need to keep re-estimating precise conditions from the distant past, but may only need to know a few aggregate properties about its history.

5.3 Generative Models and Probabilistic Programming Languages

Generative models are closely related to graphical models, but attempt to "hide" the graph and the inference engine details behind the user-friendly interface of a so-called probabilistic programming language (PPL). We can explain this by an example (Figure 19) from the work of Tenenbaum and collegues.[40] The desired task is the recognition of handwritten characters, here drawn from a fictitious large alphabet with more than a thousand different characters. This is a task that Deep Learning is highly capable of doing if it is given a large data set of handwritten characters (in pixel format), labeled by the correct character identification.[41] But what if instead we are given only a single example of each different character?

A high-level view is that we want to replace the (say) thousand-dimensional space of all (say) 32 by 32 pixel images with the much lower-dimensional space of likely handwritten characters, based on some a priori knowledge of the latter. We do this by writing, in the provided PPL, a *forward or generative model* that generates "random" handwritten characters. That is, it chooses, from some probability distribution, a number of strokes to make; chooses their shapes

[40] Brenden M. Lake, Ruslan Salakhutdinov, Joshua B. Tenenbaum, "Human-level concept learning through probabilistic program induction," Science, 350:1332 (2015).
[41] In fact, digit recognition (0-9) by a neural network has been used by the U.S. Post Office since the 1990s.

and orientations from a limited set of possibilities; and chooses where to place them with respect to one another. The various choices of probability distributions are the Bayes priors of the generative model. Without informative data, the output of the purely forward model, each time it is run, is the pixel image of a random handwritten character from a random hypothetical alphabet.

But now, as discussed in Section 5.2 above, we can reverse the process and do inference. *Given* a pixel image, we can *infer* probabilistically the values of all the random variables in the forward model. These lie in the desired lower-dimensional space sufficiently robustly that we can classify new characters in a test set based on the single exemplars of the training set. Or, given the constraints provided by the single training exemplar, we can run the forward model many times and produce many different examples of the same character (Figure 20). If we wanted, we could then train a DNN on this synthetic data, and then use that DNN as the actual recognition engine.

Generative models challenge the Big Data / Deep Learning dogma that data representations need only be flat, not structured so as to encode known properties of the data. Instead of relying on the DNN to learn from many examples how characters are made up of strokes (etc.), we specify this information in the form of a generative model and its Bayes priors. This is, in effect, a highly structured data representation. With enough labeled data, DL by itself might succeed with a flat representation; but, leveraging its richer representation, the PPL (or its underlying graphical model) can achieve success with much less data.

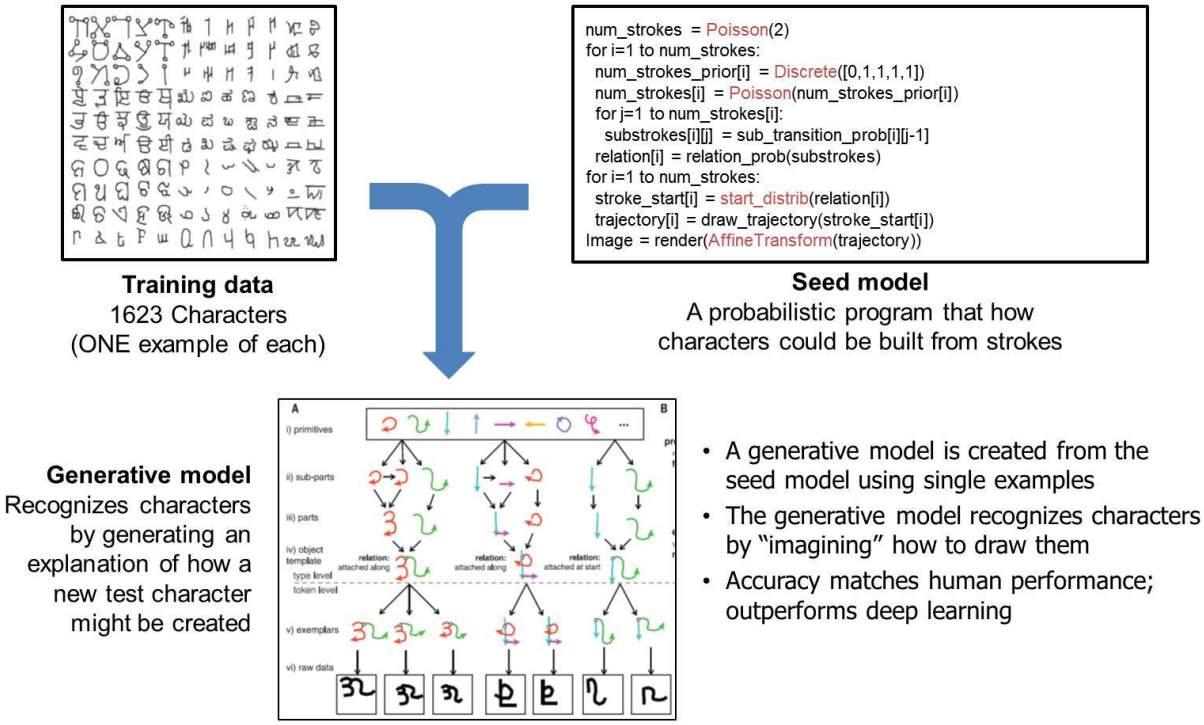

Figure 19. Application of a generative model to the recognition of (fictitious) alphabet characters given only a single training example of each character. (Source: Lake et al., in DARPA briefing).

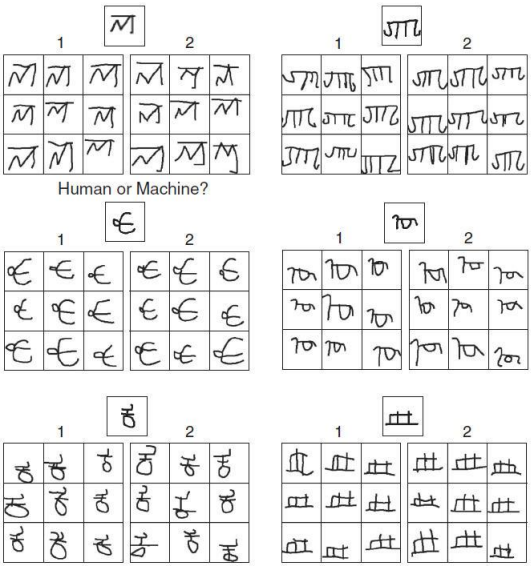

Figure 20. Given the single examples shown, human and machine each draw nine examples of the character. In each pane, human is shown randomly on either the left or the right, illustrating that the generative model captures information virtually indistinguishable from the human. (Source: Lake et al.)

Work by Stuart Russell and colleagues illustrates the practical utility of generative models in a defense-related problem. The Comprehensive Nuclear Test Ban Treaty Organization (CTBTO) maintains a global network of 147 seismic stations that constantly monitor for treaty violating underground nuclear tests. The recorded seismograms are of course vastly dominated by naturally occurring earthquakes—nuclear tests are very rare events. The NET-VISA system encodes what natural seismic events should look like as a generative model written in a probabilistic programming language, and does inference to see if the imputed parameters of any particular seismic event fall outside of the range possible for natural events. NET-VISA is predicted to achieve an approximately 60% reduction in the number of missed nuclear events as compared with the previous human labor-intensive system. [42]

5.4 Hybrid Architectures

There exist many examples of hybrid architectures in which DNNs are only one of several components. We consider this a fruitful research area. A few examples:

- AlphaGo[43] uses a DNN as something like a subroutine for the rapid evaluation of a configuration's merit, but it then explicitly explores a cleverly pruned game tree of possible moves, sometimes to considerable depth. The strategy is reminiscent of Daniel Kahneman and colleagues' understanding of the human brain as having two distinguishable thought systems, one fast, approximate, and subconscious; the other

[42] http://people.eecs.berkeley.edu/~russell/papers/bssa-netvisa.pdf
[43] http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

slower, more conscious, and more calculating.[44] It is a matter of interest whether this kind of duality is fundamental to the practical solution of some classes of problems.

- Generative adversarial networks (GANs) are proving themselves as one powerful architecture. Here two DNNs are trained simultaneously: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game.[45] In more complicated architectures, layers of G and D separately interact. These techniques are said to be widely used by Facebook. [46]
- Autoencoders, that is, back-to-back DNNs in an "encoder-decoder" configuration (Section 3.7.3), have been used successfully to learn to rotate classes of three-dimensional objects, given only a single two-dimensional view of each object.[47] This involves augmenting the features coming out of the "encoder" by a new posited parameter, the rotation angle, and then training for overall consistency of both encoder and decoder with the data.
- So-called "wide-and-deep" architectures have proved useful in applications such as recommender systems and search.[48] These combine a DNN that returns "good guesses" for answers to (say) a search query—the deep—with boolean logic search trees that can find exact matches across broad data—the wide.
- DNNs, CNNs, and Long-Short Term Memory Recurrent Neural Networks (LSTMs) are used in a combined architecture for speech recognition.[49] (See Section 3.7.4.)
- The nonlinearity between layers of a DNN can be replaced by references to memories of various sorts.

These examples by no means survey the field. They simply indicate that "pure" DL is not necessarily the best solution to all problems in AI. Rather, it seems likely that—as some of the more dogmatic aspects of Big Data / Deep Learning gradually lose currency—DNNs will remain as powerful architectural elements in the advancing disciplines of AI.

5.5 What Is On the Sidelines

Far from showing the kind of rapid progress that we have discussed above, some sub-fields of AI seem to be left behind. It is never possible to know with certainty that a sub-field will not acquire new life in the future. Indeed, that is exactly what happened as neural nets (long a backwater of AI) transformed into the hot subject of Deep Learning. Nevertheless, we here provide JASON's collective (if subjective) judgment that certain areas of AI are not poised to leap forward at present:

- Capturing expert judgment (other than for labeling big data sets)
- Physical and deterministic modeling (for example, physics-based computer vision)

---

[44] Daniel Kahneman, Thinking Fast and Slow (Farrar, Straus and Giroux, 2011)
[45] https://arxiv.org/abs/1406.2661
[46] http://www.scientificamerican.com/article/when-will-computers-have-common-sense-ask-facebook/
[47] https://arxiv.org/abs/1601.00706
[48] https://research.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
[49] http://research.google.com/pubs/archive/43455.pdf

- Direct computation on complex symbolic data representations (for example, SAT-solving approaches to cognitive systems[50])
- Purpose-designed and rule-based systems designed for error-free response (for example, the "classical" side of control theory)
- Biomimetic cognitive systems

We recognize that a lively argument could be had over any of the above judgments.

[50] By contrast, SAT-solving for formal validation of software or hardware is a lively and important sub-field.

6 DNNs FROM A HARDWARE PERSPECTIVE

6.1 Evolution of DNNs

It is instructive to examine the evolution of DNNs over the last several years. Table 1 shows key properties of the networks that won the ImageNet competition from 2012 through 2015. AlexNet[51] is a 7-layer network with 5 convolutional and 2 fully-connected layers that won the ImageNet 2012 competition. In 2013, VGG19[52] (a larger variant of VGG16) won the competition with a 19-layer network that included 2 fully-connected layers.

Table 1. Evolution of DNNs for ImageNet.

Network	Year	Conv Layers	FC Layers	Parameters	Activations	Operations
AlexNet	2012	5	2	6.1×10^7	8.1×10^5	1.5×10^9
VGG16	2013	13	3	1.4×10^8	1.4×10^7	3.1×10^{10}
GoogLeNet	2014	22	0	7.0×10^6	4.7×10^6	3.2×10^9
ResNet	2015	152	0	6.0×10^7	2.2×10^7	2.2×10^{10}

A major qualitative change came in 2014 when GoogLeNet[53] won the competition with a network containing only convolutional layers. The network used a unique architecture consisting of layers of inception modules (Figure 21) each of which included series and parallel combinations of 1 × 1, 3 × 3, and 5 × 5 convolutions. A 1 × 1 stage is used before each 3 × 3 or 5 × 5 stage to reduce the number of channels before the more expensive convolution. In many respects, the 1 × 1 convolutions take the place of the fully-connected layers. They are, in fact, a fully-connected layer per pixel. By adopting this fully-convolutional organization, GoogLeNet exceeded the accuracy of VGG16 with 20× fewer parameters and 10× fewer operations.

ResNet[54] won the competition in 2015 with an extremely deep (152-layer) network that added residual or bypass connections around each layer. The bypass connections facilitate training deeper networks by avoiding the problem of vanishing gradients.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[53] Christian Szegedy, et al. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.
[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

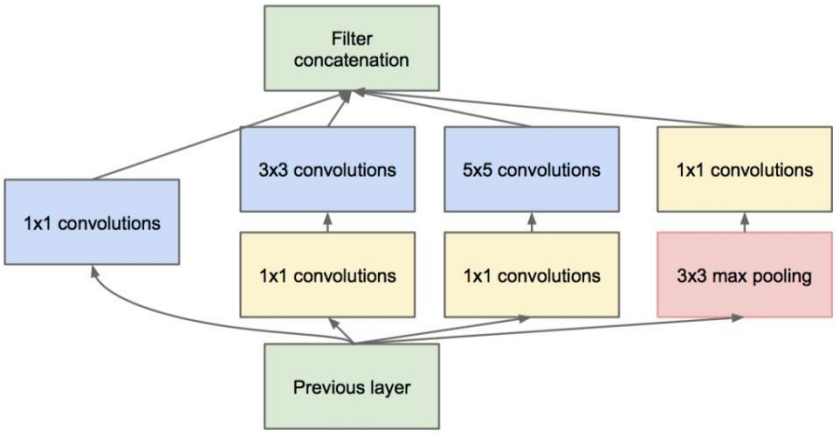

Figure 21. An Inception module from GoogLeNet. Each module consist of parallel 1x1, 3x3, and 5x5 convolutions stages. A 1x1 stage with c > k is used before the larger convolutions to reduce dimensionality.

Several important trends are clearly visible from the data in the table:

Increasing Depth: The depth of the networks is increasing — from 7 layers in 2012 to 152 layers in 2015. This is because deeper networks give better results. The depth of *mainstream* networks has been increasing somewhat slower than this with 20-50-layer networks common today.

Improved Efficiency: Newer networks are doing more with fewer parameters. GoogLeNet has an order of magnitude fewer parameters than AlexNet or VGG16 but gives better accuracy. This increase in efficiency is largely due to the move to fully-convolutional networks.

Fully-Convolutional Networks: Starting with GoogLeNet in 2014, these networks no longer have fully-connected layers. Instead all layers are convolutional. However, these newer networks contain many 1 × 1 filters which are the equivalent of a fully-connected layer *per pixel*. Moving to fully-convolutional networks gives a net reduction in the size of the model required to achieve a given accuracy. Fully-connected layers remain important for other types of DNNs.

Smaller Convolutions: As networks have gotten deeper, the size of the convolutions have been reduced. AlexNet included an 11 × 11 convolution in the first layer and a 5 × 5 convolution in the second layer. VGGNet moved to 3 × 3 convolutions. Better performance is achieved with more layers of smaller convolutions to enable more non-linearity in the filters.

The evolution of CNNs for ImageNet illustrates the triad of network scaling. To improve accuracy the size of the training set, the capacity of the network, and the performance of training hardware must all improve together. Network capacity and training set size are closely coupled. If too large a network is trained on a small data set, the network *overfitting* will occur. The network will give good results for the training set, but will generalize poorly to other input data. On the other hand, too small a network will have inadequate capacity to learn all of the information in a large data set. If a network gives the same accuracy on half the training set that it gets on the full training set, it is too small.

While the ImageNet data set has held constant at 1.2 million images, network designers are able to get the effect of a larger data set by using *data augmentation* (Section 3.7.2). Each image in the original training set is used to generate many (sometimes hundreds) of *augmented* images by cropping, mirroring, warping, adjusting white balance and color balance, etc. Training on augmented data has been shown to improve performance on the test set. Of course increasing the size of the training set also increases training time.

The size of the network and the size of the data set are ultimately limited by the computing power available for training. Today's DNNs are sized to be trainable in no more than two weeks on small clusters of 8-16 GPUs. If the network size and training set size both doubled, the training time would quadruple. To stay within the two-week threshold on training time, the performance of computer hardware would need to quadruple to enable this doubling of network capacity.

While the architecture of networks has evolved considerably, they continue to be built from the same basic sets of primitives. The computation of a network is dominated by convolutions and matrix-vector multiplies (matrix-matrix multiplies, for batched computations).

6.2 DNN Computations

Two main types of computation are performed on neural networks: *training* and *inference*. During the training process the weights of each connection in the network are *learned*—by iteratively applying labeled inputs and adjusting the weights to improve accuracy. Once learned these weights can be used for inference by applying an input to the network and computing a set of output values.

6.2.1 Training

The training process is used to compute the weights of a neural network from a labeled training set of input-output pairs[55]. During the training process each element (e.g., image) in the training set is applied to the network, an output is produced, and a *loss function* is computed by comparing the computed output to the correct output. The weights are then updated based on the loss function (see Section 3.4). Using stochastic gradient descent (Section 3.6.4), a partial derivative of the loss function with respect to each weight is computed through *back propagation*. Running the network backwards computes the partial derivative of the loss function with respect to each activation — applying the chain rule repeatedly. The partial derivative with respect to the weights can then be computed from the partial with respect to the output activations and the original input activations.

To improve efficiency, training is typically performed on batches of 2^6 to 2^{10} images at a time. Operating on batches converts the matrix-vector product to a matrix-matrix product and increases the re-use of weights — reducing the demand on memory bandwidth.

To avoid having small weight updates become lost in quantization noise, most networks today are trained using 32-bit floating-point representations of weights and activations. There is some

[55] It is also possible to train a network using an unlabeled training set by using an autoencoder or by using reinforcement learning. However these methods are beyond the scope of this section.

evidence that 16-bit floating-point is adequate for training, particularly if accumulation is done at 32-bit precision and stochastic rounding is employed.

Training is a computationally intensive process. Training sets are typically large, containing 10^6 to 10^8 samples, and they must be applied multiple times, often 10^3 or more (called training epochs) before the network converges. Processing each training input through the forward computation and back propagation may take $10^9 - 10^{10}$ operations. Thus training a network can easily take 10^{18} to 10^{21} operations. Even on a 10TF (10^{13} FLOPS) GPU, training can take up to 10^5 to 10^8 seconds (1 day to 3 years).

The memory footprint for each training batch can be quite large. 10^6 to 10^8 activations for 10^3 samples need to be retained, resulting in a memory footprint of 10^9 to 10^{11} 32-bit words or 4 to 400 GB. This large footprint requires that activations be stored in DRAM during training. The footprint is far too large for an on-chip SRAM.

Network accuracy improves with larger networks and training sets. However, the size of networks and training sets used today is largely limited by the computational resources available for training.

6.2.2 Inference

Once a network has been trained, it can be used for inference. For each input stimulus applied to the network, a set of output values is computed. For example, an image is applied to the network and a set of output values that classifies the image and/or detects an object in the image is computed. For example, a one of N output can specify what class of object is in the image for a classifier network. A bounding box with label may be output for a detection network.

Compared to training, inference is a much simpler computation for three reasons. First, good accuracy can be achieved representing weights and activations with 8-bit fixed-point numbers. Second, each layer of activations can be discarded as soon as the next layer is computed, resulting in a much smaller memory footprint. Finally, the back-propagation and weight update steps are not needed. This cuts the computation per image by a factor of approximately three.

While some applications of DNNs permit inference calculations to be batched, others are latency sensitive and require each input stimulus to be processed as soon as it is received.

6.2.3 Compute and Memory Requirements for DNNs

The convolutional layers of a DNN are compute, not memory bound. The number of operations used to compute one convolutional layer (for both the forward and back-propagation steps) is equal to the size of the output activation A_{i+1} ($k \times x \times y$) multiplied by the size of the convolutional kernel used to compute each point in the output activation ($c \times r \times s$). That is, the computation is a six-dimension loop that iterates over x, y, c, k, r, and s. For example, the *conv4_3* stage of VGG16 computes a $28 \times 28 \times 512 = 4.0 \times 10^5$ point output activation using a $512 \times 3 \times 3 = 4.6 \times 10^3$ point convolutional kernel to compute each activation point. Thus, this layer requires 1.9×10^9 multiply-adds or 3.7×10^9 FLOPS. The memory footprint for this computation is 4.0×10^5 activations and 2.4×10^6 parameters. Thus, this computation performs over 10^3 FLOPS for each word read from memory — even without batching.

In contrast, the fully-connected layers of a DNN may be memory bound, particularly for small batch sizes. The number of operations required to compute on fully-connected layer is equal to

the size of the parameter matrix W_i, which is equal to the product of the number of input and output activations. For example, the *fc7* layer of VGG16 has 4K input activations and 4K output activations. Thus W_i contains 16M parameters, and 16M multiply-adds (32M FLOPS) are required to compute the layer. Compared to this relatively modest amount of computation, the memory footprint is huge, with 16M parameters.

Without batching a word has to be read from memory for every multiply-add. With batching, the re-use is equal to the batch size.

6.3 Hardware for DNNs

As we have seen from the previous section, both training and performing inference on DNNs require performing dense linear algebra operations—convolutions and matrix-vector multiplies. With the exception of unbatched fully-connected layers (which are memory bound), the computation is arithmetic bound. Training computations are largely performed in 32-bit floating point while inference computations are mostly performed in 8-bit fixed point.

These computations can be performed on CPUs, GPUs or fixed-function accelerators. For each platform what matters is the raw performance (multiply-adds/second) and the efficiency (multiply-adds/J). For unbatched fully-connected layers, the memory bandwidth (bytes/s) and efficiency (bytes/J) also matter. Because one can easily parallel multiple units to get higher performance, the efficiency numbers are the most important.

The relationship between network architecture and hardware architecture is bi-directional. Future hardware is being driven by observed trends in networks, to meet the anticipated demand. On the other hand, new networks are strongly influenced by what runs well on existing hardware. In particular, the size of a network and its training set are limited primarily by available training hardware.

Table 2. The efficiency (ops/J) of CPU, CPU with vector extension, GPU, and accelerator on 32-b floating-point and 8-b fixed point operations. The first two columns are *peak* performance. The third column is sustained performance on AlexNet. The final column expresses the sustained performance as a fraction of peak.

Type	8b OPS/J	32b FLOPS/J	AlexNet	% Peak
Scalar CPU	1×10^9	1×10^9		
CPU Vector Extension	3.6×10^{10}	9×10^9	1×10^9	11 %
GPU (GP100)	1.4×10^{11}	3.5×10^{10}	1.4×10^{10}	40 %
Accelerator	8×10^{11}	1.2×10^{11}		

The first two columns of Table 2 compare the theoretical maximum efficiency of four hardware platforms on 32-b floating-point and 8-b fixed point arithmetic. This theoretical maximum does not include the energy for referencing memory, index calculations, control overhead, etc. Thus, no real platform will achieve these numbers. They are an upper bound on what is achievable.

A CPU executing scalar instructions consumes about 1 nJ of energy per instruction in its complex, dynamically-scheduled pipeline. This overhead energy swamps the 4 pJ for the floating-point operation, the 4 pJ for a 32-b SRAM read, or the 0.3 pJ for an 8b integer multiply. The efficiency is 1 GOP/J — almost all of it consumed in instruction scheduling overhead.

CPUs attempt to combat instruction overhead by adding SIMD (single-instruction, multiple-data) or vector extensions. The latest Intel processors, for example, support the AVX-512 extension that performs 512-bit wide vector operations: 16 32-bit operations or 64 8-bit operations are performed in parallel. It is difficult to write code to exploit the AVX extension. However, highly tuned libraries, such as the Intel MKL library, are available to perform the convolutions and matrix multiplies needed for DNNs on AVX. The AVX parallel operations amortize the 1nJ instruction scheduling overhead resulting in theoretical efficiencies of 9 GFLOPS/J for 32-bit FP and 36 GOPS/J for 8-bit fixed point. Using the Intel MKL libraries, the E5-2697-E4 sustains 2.1 GFLOPS/J on AlexNet or 11% of the peak theoretical efficiency.

GPUs have very simple execution pipelines and amortize instruction overhead over 32 processors using a SIMT (single-instruction multiple-thread) architecture. As a result, they have very high execution efficency. The NVIDIA GP100 (Pascal) GPU has peak efficiency of 35 GFLOPS/J on 32-bit floating point and 140 GOPs/J on 8-bit fixed point. The GPU achieves 40% of peak, 14 GFLOPS/J on AlexNet.

For training, raw performance can be as important as efficiency because network and training set size are limited by training speed. Table 3 compares the training rate of three platforms: a CPU, a single GPU, and an 8-GPU cluster (DGX-1). The 18-core Xeon CPU can process 100 images per second on AlexNet. The single GPU provides 29 times this performance, and the cluster of 8 GPUs provides 190 times this performance.

Table 3. The raw performance (in AlexNet images/s) of three Deep Learning training platforms.

Platform	AlexNet images/s
CPU (E5-2697-v4)	100
GPU (GP100)	2,938
DGX-1 (8 × GP100)	19,053

The numbers in Table 3 show that small numbers of GPUs achieve nearly linear speedup on training of DNNs. This allows even larger networks to be trained in an acceptable amount of time. Training on 8-16 GPUs is common, and Baidu has reported linear scaling up to 128 GPUs[56]. Earlier results from Google also report parallelizing training[57].

The ultimate in efficiency is a fixed-function accelerator. By hard-wiring a particular function, most energy is consumed by the arithmetic units themselves (4pJ/32-b FLOP or 0.3pJ/8-b op) and small SRAMs used to stage data into the execution units (1pJ/byte). Note that for 8b integer operations, this memory read takes more energy than a multiply. A real accelerator will not achieve these numbers because additional energy is required to read data from DRAM, control the pipelines, and perform other functions. We examine the efficiency of some real accelerators in Section 6.3.1.

[56] Dario Amodei, et al. Deep speech 2: End-to-end speech recognition in English and Mandarin. arXiv preprint arXiv:1512.02595, 2015.

[57] Jeffrey Dean, et al. Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012.

From Table 2 and Table 3 we see that:

1. Of COTS hardware, GPUs are significantly more efficient than CPUs at executing DNNs. On AlexNet, an NVIDIA P100 has 14.2 times the sustained FLOPS/J than a Core E5.
2. GPUs also offer substantially better total performance. This enables training of larger networks on larger data sets. A single P100 GPU executes AlexNet 29 times faster than an 18-core server CPU. A cluster of 8 P100 GPUs operates 190 times faster.
3. Accelerators could potentially be substantially more efficient than GPUs, particularly at inference which uses 8b operations. We examine accelerators in the next section.

6.3.1 Accelerators for DNNs

While GPUs provide impressive performance and efficiency executing DNNs, as Table 2 suggests, fixed-function accelerators offer the potential for even greater performance and efficiency. One of the first efforts in building special purpose hardware for DNNs is the DianNao family of chips that were jointly developed by the Chinese Academy of Sciences and INRIA[58] [59] [60]. This family of accelerators is based on a core, illustrated in Figure 22, that performs dot product operations. The original DianNao chip had 16-units that each performed 16-point, 16-bit dot-product operations, for a total of 256 multiply-accumulates per cycle. In 65nm CMOS this chip dissipated 485mW at 980MHz for an efficiency of approximately 1 TOPS/J, not counting memory energy.

The DianNao chip fetches all activations and weights from off-chip DRAM. With an access energy of 320 pJ/word (compared to 1 pJ/MAC) external memory access completely dominates system energy as shown in Figure 23. The DianNao core has 64-entry input buffers for both weights and activations. However, these buffers are far too small to get sufficient reuse to amortize the very large cost of reading data from external memory. The result is an actual efficiency of 42 GOPS/J.

[58] Tianshi Chen, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In ACM Sigplan Notices, volume 49, pages 269–284. ACM, 2014.
[59] Yunji Chen, et al. Dadiannao: A machine-learning supercomputer. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 609–622. IEEE Computer Society, 2014.
[60] Zidong Du, et al. Shidiannao: shifting vision processing closer to the sensor. In ACM SIGARCH Computer Architecture News, volume 43, pages 92–104. ACM, 2015.

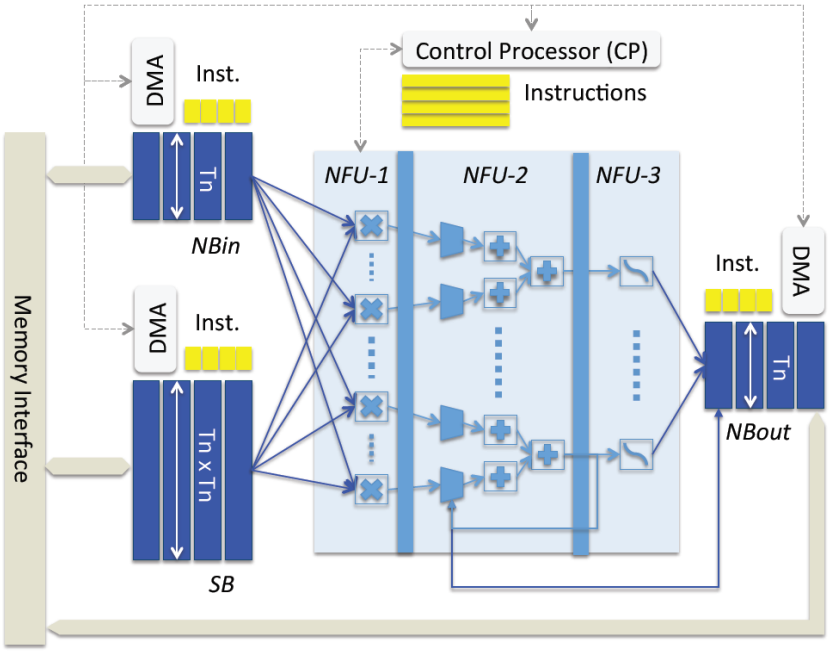

Figure 22. The core of the Dian Nao DNN accelerator performs a 16-point, 16-bit fixed-point dot product operation followed by a programmable non-linear function. Input and output buffers facilitate operand reuse.

To address the issue of high memory energy the DaDianNao chip was developed with embedded DRAM (eDRAM)[61]. Each DaDianNao chip contains 36MB of eDRAM. This memory is partitioned into 2MB arrays in each of 16 tiles and a 4MB central array. By eliminating the energy required to access external DRAM, DaDianNao achieves an efficiency of 350 GOPS/J.

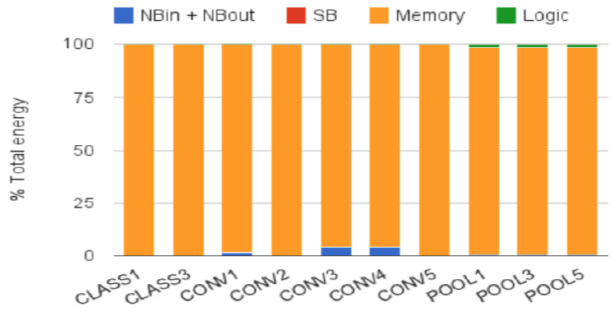

Figure 23. Because it fetches all data from external DRAM. the power dissipation of DianNao is completely dominated by memory fetch.

[61] Yunji Chen, et al. Dadiannao: A machine- learning supercomputer. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 609–622. IEEE Computer Society, 2014.

In an alternate approach to address the issue of high memory energy, a group at NVIDIA and Stanford has developed the efficient inference engine (EIE)[62]. This design leverages network compression techniques that first prune the network, eliminating unnecessary connections[63] and then compresses the remaining weights using *trained quantization*.[64] This process reduces the size of the network by 240× allowing it to fit in a modest-sized on-chip SRAM which can be accessed for 5 pJ per 32b word rather than 640 pJ for off-chip DRAM.

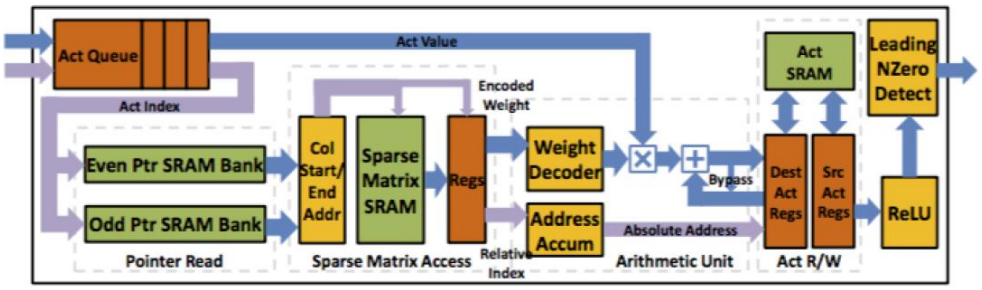

Figure 24. Block diagram of the efficient inference engine (EIE). EIE keeps the weights in a highly-compressed form (compressed 240×) by pruning the network and coding the non-zero weights using trained quantization. The special-purpose hardware is tailored to operate efficiently on this sparse, encoded representation.

The EIE is designed to exploit the inherent sparsity of DNNs. Both the weights and the activations of DNNs are sparse. The weights (connections) can be pruned to 10% density for fully-connected layers and to 30% density for convolutional layers with no loss of accuracy.[63] With ReLU non-linear units (which convert negative values to zero), a large fraction of activations are zero as well. For a typical layer 30% of the activations are non-zero. Thus, for a typical fully-connected layer, with 30% activation density and 10% weight density, only one operation in 30 of the dense computation actually needs to be performed. For a convolutional layer, only one operation in 11 needs to be performed. The sparsity of the weights is static, while the sparsity of the activations is dynamic, varying from one input to another.

The EIE is a tiled design intended to operate on compressed, sparse fully-connected layers. A sparse adjacency matrix for the layer is row interleaved across the tiles. Input and output activations are also interleaved. Except for the broadcast of non-zero input activations, all operations are local to a tile. On a wide range of networks, near linear speedup is demonstrated up to 256 tiles. The block diagram of one EIE tile is shown in Figure 24. The hardware is tailored to efficiently walk the compressed sparse column matrix representation and to decompress the weights. Performing these operations on COTS hardware is inefficient.

---

[62] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. arXiv preprint arXiv:1602.01528, 2016.
[63] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, pages 1135–1143, 2015.
[64] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. ICLR, 2015.

Eyeriss, jointly developed by MIT and NVIDIA, exploits activation sparsity. It also uses a large buffer memory and a spatial-array architecture to minimize DRAM energy.[65] [66] This allows it to achieve 170 GOPS/J even though it is fetching all activations and weights from off-chip DRAM. It is able to get sufficient on-chip re-use to amortize the high DRAM access energy.

Table 4. The efficiency of fixed-function accelerators for DNNs.

Accelerator	OPS/J
Theoretical Maximum	8×10^{11}
DianNao	4.2×10^{10}
Eyeriss	1.7×10^{11}
DaDianNao	3.5×10^{11}
EIE	3.5×10^{11}
EIE dense equiv.	1.1×10^{13}

Table 4 compares the efficiency of a number of fixed-function DNN accelerators to the theoretical maximum efficiency. DianNao has an efficiency of 42 GOPS/J. This is substantially worse than GPU performance on inference (8b fixed point). Accelerators may not be more efficient than programmable engines if they do not efficiently deal with sources of overhead — like memory energy.

Eyeriss improves on the efficiency of DianNao by using a large enough input buffer (108KB) to get sufficient re-use to amortize the high energy of fetching weights and activations from DRAM. It also uses a spatial array to facilitate re-use between processing elements and gates off execution units when activations are zero to exploit sparsity. The result is an efficiency of 170 GOPS/J. DaDianNao doubles this efficiency, to 350 GOPS/J, by storing all weights and activations in on-chip eDRAM.

EIE also stores all weights and activations in on-chip RAM, by using pruning and compression so that weights and activations fit in on-chip SRAM. It matches the efficiency of DaDianNao at 350 GOPS/J. For EIE, however, this is efficiency on a *sparse* computation. By exploiting sparsity in both activations (3×) and weights (10×) EIE can evaluate a network with 30× fewer operations. To match the efficiency of EIE, a dense accelerator like DaDianNao would need to have an efficiency of 11 TOPS/J. This *dense equivalent* efficiency is the last line of Table 4. Note that GPUs and CPU SIMD extensions are relatively inefficient on sparse calculations, so inference on sparse networks is one area where fixed-function accelerators can achieve a significant advantage over programmable engines.

There are a few commercial offerings of accelerators for DNNs. Google announced a tensor processing unit (TPU) to accelerate inference on DNNs.[67] [68] However no details of the device

[65] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In 2016 IEEE International Solid-State Circuits Conference (ISSCC), pages 262–263. IEEE, 2016.
[66] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. 2016.
[67] Karl Freund. Google's tpu chip creates more questions than answers. Forbes, May 2016. http://www.forbes.com/sites/moorinsights/2016/05/26/googles-tpu-chip-creates-more-questions-than-answers
[68] Norm Jouppi. Google supercharges machine learning tasks with tpu custom chip. Google Cloud Platform Blog, May 2016.

have been disclosed other than that it is 10× more energy efficient than an unspecified alternative.

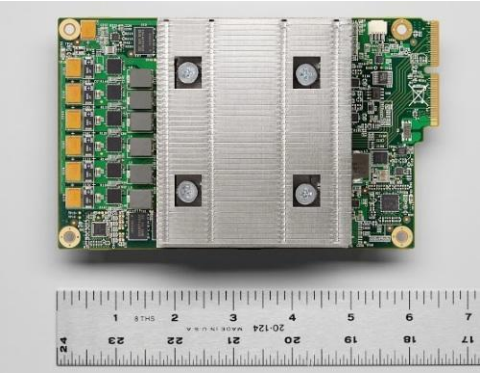

Figure 25. A photo of Google's tensor processing unit (TPU). Few details of this chip have been released. All that is known is that it gives 10× the efficiency as some undisclosed alternative on inference.

Movidius makes a *vision processing unit* (VPU) that accelerates convolutional networks for computer vision.[69] Their Myriad 2 chip has 2 TFLOPS of 16-bit floating-point performance.

With the end of Dennard Scaling[70] and with it the performance scaling popularly referred to as Moore's Law[71], we can no longer expect large improvements in performance due to improvements in semiconductor process. This is true both for general-purpose CPUs and GPUs and for fixed-function DNN accelerators. Each generation of semiconductor technology, which reduces linear dimensions by 30%, is expect to give about a 20% improvement in OPS/J. This is in contrast with a 180% improvement when Dennard scaling was in effect.

We expect most future improvements in performance and efficiency to come from improvements in architecture and more optimized networks, rather than from process technology. GPUs are now being optimized for DNN training and inference. For example, the latest generation of NVIDIA GPUs has support for 16b (half-precision) floating point and 8b integer operations. Special instructions to accelerate the inner loops of DNN algorithms are also starting to appear in newer GPUs. We expect this continued evolution of GPUs to eliminate essentially all of the gap between GPUs and fixed-function hardware for training, and much of the gap for inference.

At the same time, we expect more fixed-function accelerators for DNN inference to appear. This is driven by the need to embed DNNs in small embedded devices from video cameras to appliances.

[69] R. Colin Johnson. Movidius vision processing unit enters 2nd generation. EE Times, July 2014.
[70] Giorgio Baccarani, Matthew R Wordeman, and Robert H Dennard. Generalized scaling theory and its application to a 1/4 micrometer mosfet design. IEEE Transactions on Electron Devices, 31(4):452–462, 1984.
[71] Hadi Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In Computer Architecture (ISCA), 2011 38th Annual International Symposium on, pages 365–376. IEEE, 2011.

6.4 Optimizations

One can also improve the speed and efficiency of training and inference on DNNs by optimizing the networks themselves. Reducing the complexity of the network, without sacrificing accuracy, has the same advantage as running on faster hardware. The two are complementary. Additional gains can be achieved by running a less complex network on more efficient hardware.

6.4.1 Compact Networks

A more efficient network design can achieve equivalent accuracy with fewer parameters and fewer operations. GoogLeNet[53] is an example of an efficient network design. By using a fully convolutional network, by making the network very deep (22 layers), and by using inception units (Figure 21) that use 1 × 1 convolutions to reduce dimensionality before applying 3 × 3 convolutions, GoogLeNet exceeds the accuracy of VGG16 while using 10× fewer operations and 6× fewer parameters.

SqueezeNet takes a similar, if slightly simpler, approach.[72] SqueezeNet is composed of alternating layers of squeeze and expand modules. A 1 × 1 convolution with more input channels than output channels squeezes the representation to fewer dimensions. A 3 × 3 convolution then expands the dimensionality. SqueezeNet achieves AlexNet accuracy with 50× fewer parameters. Combining this compact network with pruning and compression[64] results in a model that is only 470KB in size compared to 240MB for the original AlexNet.

6.4.2 Sparsity

DNNs are inherently sparse, both in activations and connections. While many DNNs are formulated as dense networks—i.e., all input activations in a fully connected layer are connected to all output activations—it has been shown that the majority of connections can be eliminated without affecting accuracy.[63] Fully connected layers can typically be pruned to 10% density, and convolutional layers can typically be pruned to 30% density. This results in a 10× and 3.3× reduction in the number of weights needed to represent a network. In addition, typically only 30% of activations are non-zero for any given input. Exploiting both the static sparsity of weights and the dynamic sparsity of activations reduces the amount of work to be performed by 33× for fully-connected layers and by 11× for convolutional layers.

However, exploiting sparsity also turns evaluation of the network into an irregular computation. The data-parallel hardware in GPUs and the SIMD extensions of CPUs are poorly matched to this irregular computation. Structured pruning[73] has been suggested as a method to reduce connections while preserving regularity. Alternatively, accelerators like EIE can efficiently walk sparse-matrix data structures.[62]

---

[72] Forrest N Iandola, et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size. arXiv preprint arXiv:1602.07360, 2016.

[73] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. arXiv preprint arXiv:1512.08571, 2015.

6.4.3 Attention Networks

A conventional CNN performs the same computation over an entire image, for example, from an HD (1920 × 1080) sensor. Much of a typical image is often empty space, sky, ground, etc., with no objects of interest. It is wasteful to run complex feature detectors and object detectors over obviously empty space.

CNNs can be made much more efficient by using a simple network to detect regions of interest (ROIs). An *attention mechanism* then directs a more complex network to direct its attention to these ROIs — processing only these areas of the image and ignoring the rest.

The R-CNN family of networks[74] [75] [76] use an attention mechanism. A relatively simple *generic object detection* network is used to identify regions of interest. A more complex classifier network and per class detection network is then applied to each region. In a similar manner[77] Xiao et al. apply attention to the classification problem. It first uses a filter network to propose ROIs. These ROIs are then input to a classifier network.

[74] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 580–587, 2014.
[75] Ross Girshick. Fast r-cnn. In Proceedings of the IEEE International Con- ference on Computer Vision, pages 1440–1448, 2015.
[76] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Ad- vances in neural information processing systems, pages 91–99, 2015.
[77] Tianjun Xiao, et al. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 842–850, 2015.

7 SOME CONSIDERATIONS SPECIFIC TO DOD

By now it should be clear that there is much potential value for the Department of Defense incorporating AI technologies into DoD systems. While much discussion in the AI community focuses on research directions and opportunities, the purpose of this section is different. Taking for granted existing and near-term AI technologies, how might the DoD use them?

In this context, many topics in the public discussion of AI become irrelevant. For example, displacement of soldiers' jobs by technology is a benefit, not an economic or social harm.

AI technologies make available more powerful or more flexible algorithms than conventional software engineering methodologies. Still, these have to be engineered into functioning systems. Satisfying 80% of the need is a good start, and sometimes sufficient for the commercial sector, but for many DoD applications, the necessary remaining 20% needs to be provided too. This simple fact has restrained the spread of AI in higher consequence aspects of everyday life, and internalizing it will help avoid failures.

For the DoD, AI provides two sorts of opportunities. 1. AI technologies might make existing tasks simpler, more reliable, or more efficient. Or, 2. AI technologies might be used to introduce wholly new capabilities. Another dichotomy is substitutive, where AI replaces people, or complementary, where AI improves or helps. But these points are true of all automation. AI techniques provide new tools to help accomplish these goals. Using them is engineering, albeit advanced engineering.

As an example that demonstrates advantages of the new technology, consider automatic target recognition, a problem that has been intensively worked on for decades. At one time careful models (real or virtual) were built and measured and the data from each model were hand coded into the algorithm. It was difficult to compute the possibility of confusion with non-targets because each would have had to be precisely modeled too. With machine learning, one can take lots of pictures of targets and of non-targets and use a clustering algorithm. It is not hard to understand the classification errors in the resulting model.

A bigger difference, however, is the much smaller amount of human intervention and hand coding that is needed compared to older techniques. Presented with new targets, or non-targets, the model can just be retrained. The algorithms find the decision boundaries automatically. Deep learning algorithms may have even better performance, although then understanding classification errors requires different techniques. The ability to upgrade the algorithm easily is an advantage. Taking advantage, however, has implications for how the algorithm is embedded in systems. When changing the target recognition algorithm was a big effort, systems were upgraded only occasionally. To exploit the advantages of the new technology the systems need to be easily upgradable, and that comes with its own set of issues and new technologies. (In private life one used to get very occasional OS upgrades on disks. Now it is all done over the network, frequently and almost transparently.)

As an example that suggests how difficult DoD's transition to increased use of AI might be, consider autonomous ground tactical vehicles. One might consider these for combat support, or just for logistics. Either would be a boon. Civilian experience shows that going down this path will require at least a decade of challenging work. The work on self-driving cars has consumed

substantial resources. After millions of hours of on-road experiments and training, performance is only now becoming acceptable in benign environments. Acceptability here refers to civilian standards of safety and trust; for military use the standards might be somewhat laxer, but the performance requirements would likely be tougher.

There are many reasons why the problem is so hard, but here are two. First, the environment is complex and challenging. The car has some sort of map. Then, if it can see lane markings, it can figure out what to do—unless there has been recent construction, in which case it must decide which are the old misleading lane markings and which are the new ones. If it is snowing, or there is a road with no lane markings, what does it do? There are engineering answers to all of this, but there are lots of situations that are only slightly unusual, but have to be handled. More complex is the interaction with other things, from other vehicles, to pedestrians, to potholes, to careless children, to windblown garbage cans. So the second reason is that it is hard to cope with the complex tree of possibilities, understanding what's going on and what to do about it. This is an area where research is needed and where research can help.

But applying that research is likely to require a lot of difficult engineering. More likely than full autonomy is some form of driver assist. In the civilian world the human interface is critical, and getting that right requires trial and error and redesign. (Smart cruise control works quite well technically, but there are people who turn it off because it annoys them in one way or another.) The military may be able to settle for good-enough. One might imagine convoys of trucks where only the first truck has a driver and the others can use the driver's plans and all the trucks' sensors. This raises the 80/20 issue: driving down the highway would work, but there are lots and lots of edge cases that have to work too. (Traffic light changes in the middle of the convoy? Driver makes a mistake and the convoy has to reverse? Flat tire? Refueling?)

In image processing the DoD can take advantage not only of existing knowledge, but also the public domain neural nets that have already been trained. It is unlikely that DOD applications want the exact same images recognized, but the results of transfer learning (Section 3.7.1) suggest that most of the neural net could be thought of as adapted to all sorts of images and can be left fixed, and that only the top few layers need to be retrained, which would be much less effort than full retraining.

Machine learning is intrinsically statistical. This fact may have implications for requirements, for testing, and for understanding the behavior of the system. Errors pose a different problem for an operational system than for a research system. A machine learning demonstration or research project starts with data purportedly representative of the real world, keeps some aside for final evaluation, and after training reports some measure of accuracy evaluated on the held out data. The measure is typically some combination of precision and recall. There are two pitfalls. The first is well-understood, that the training data may not be sufficiently representative of the real world. The second is that in the real-world, some errors are much worse than others. If these can be anticipated, then extra training data can be used to make sure these mistakes don't happen. Commercial experience suggests that it's very hard to think of all of these in advance; operational systems may well need to be changed quickly.

The implication is that requirements need to include something about error rates, and testing needs to go beyond checking that requirements are satisfied. DoD's testing of AI systems will have to probe outside the boundary of expected behavior to try to uncover unexpected weaknesses.

8 JASON FINDINGS AND RECOMMENDATIONS

8.1 Findings

The background for each of the following findings is found in the main text. Pointers are given to the relevant sections.

1. AI, both commercially derived and DoD-specific, will be integral to most future DoD systems and platforms. To be a "smart buyer" and to support its unique development needs, DoD needs to maintain a strong portfolio of 6.1 research, and a cadre of knowledgeable program officers, widely across the most rapidly advancing areas. [Section 2.2]
2. The boundary between existing AI and hoped-for AGI keeps being shifted by AI successes, and will continue to be. For DoD, AGI is at most a small part of AI's relevance to the DoD mission. [Section 2.3.2 and Chapter 0]
3. The use of AI to augment human performance is a key application area for DoD, and closer to real implementation than AGI. [Section 2.2]
4. DNNs have exceeded typical human performance on many tasks, including face recognition, object detection, and speech understanding. [Chapter 0]
5. DNN performance is dependent today on large, labeled training sets of data. Reinforcement learning (RL), transfer learning, and autoencoders can replace labeled data in special cases. [Section 3.6.5]
6. DNNs are "winning the race" today, but large Bayes models (e.g., graphical, and with PPLs) continue to be innovative and fruitful, especially for DoD-related applications. Generative Bayes models can require much less data for training. [Chapter 0]
7. Hybrids show great promise, both for (i) AI engineering at the "block level", and (ii) for richer data representations within DNN-like multilayer structures (e.g., memories, generative models, networks-of-networks). [Section 5.4]
8. COTS GPUs offer enabling performance for training DNNs. Special purpose hardware, both COTS and DoD-specific, hold the potential for even higher efficiency on inference. [Chapter 6**Error! Reference source not found.**]
9. DNNs are immature as regards the "illities", including reliability, maintainability, accountability, validation and verification, debug-ability, evolvability, fragility, attackability, and so forth. [Chapter 4]
10. Increasingly, cutting-edge AI research can be performed only by large organizations with access to key resources (e.g., large labeled datasets and large farms of GPUs) [Section 3.8]
11. Most AI models need at least periodic retraining. The acquisition process must cope with this. [Chapter 7]

8.2 Recommendations

JASON offers the following recommendations to DoD senior leadership:

1. DoD should both track (via a knowledgeable cadre) and invest in (via a 6.1 research portfolio) the most dynamic and rapidly advancing areas of AI, including, but by no means limited to DL.

2. DoD should support the development of a discipline of AI engineering, accelerating the progress of the field through Shaw's "craft" and (empirical) "commercial" stages. A particular focus should be advancing the "illities" in support of DoD missions.
3. DoD's portfolio in AGI should be modest and recognize that it is not currently a rapidly advancing area of AI. The field of human augmentation via AI is much more promising and deserves significant DoD support.
4. DoD should support the curation and labeling, for research, of its unique mission-related large data sets. Wherever possible, operational data should be saved for future research use in support of AI for DoD-unique missions.
5. DoD should create and provide centralized resources for its intramural and extramural researchers (MOSIS-like), including labeled data sets and access to large-scale GPU training platforms.
6. DoD should survey the mission space of embedded devices for potential breakthrough applications of AI, and should consider investing in special-purpose accelerators to support AI inference in embedded devices for DoD missions if such applications are identified.

APPENDIX A: Statement of Work and JASON Responses to Specific Questions Posed

The Statement of Work posed to JASON includes four specific "yes/no" questions regarding the state of development of Artificial General Intelligence. While the detailed answers to these questions are implicit in the main body of the report, we here give the actual text of the Statement of Work, and JASON's responses to the questions posed.

Statement of Work from ASD (R&E)

Objectives:

There have been notable advances in intelligent robotic systems in recent decades including development of marine survey robots, self-driving cars, home vacuum cleaners, outer planetary probes, and many others. These intelligent systems function reasonably well in highly structured, well-specified narrow domains and scenarios. However, it is generally acknowledged these systems tend to exhibit very limited robustness and adaptabilities when they are placed in unstructured, open-world, uncertain environments that are of critical importance to DoD. To build robust, versatile agents that can function in a variety of environments (physical or cyber) and situations, and capable of interacting/reacting to friendly/unfriendly intelligen.t agents, we must develop the science base for general machine intelligence or Artificial General Intelligence (AGI).

Since 1950s when Artificial Intelligence (AI) became a recognized discipline, we have tried to develop the science for building smart agents that are versatile and resilient. Even though we have had great success in building agents that perform well in structured, narrow, closed domains, the field has not yet achieved its promise. What is missing then? Most researchers agree that general intelligence is indeed knowledge acquisition, knowledge representation, reasoning, and communications with other intelligent agents.

Scope:

1. Have there been the necessary breakthroughs and framework to create structures of general intelligence for acquiring knowledge? That is access to varied sources of information, algorithms for learning, tools for knowledge engineering, and architectures for organizing the acquired knowledge in ways that are useful for inference?

JASON response: Generally speaking, no. The enormous recent progress in AI, while broadly applicable to many areas, continues to advance in a domain specific manner.

2. Have we made sufficient progress to be able to create elements of general intelligence on reasoning, planning (decision-making), and problem solving in large, uncertain, partially known, open domains?

JASON response: The breakout technologies associated with Big Data / Deep Learning are applicable to some partially known, open domains, most notably self-driving vehicles in urban settings. However, this progress is not the result of the creation of elements of general intelligence.

3. Do we have sufficient level of understanding in cognitive neuroscience of social cognition to enable us

a. To develop computational architectures and interfaces to engender trust in the users?

JASON response: No.

b. To allow intelligent systems to represent goals, plans and perspective of the users.

JASON response: There is considerable potential in this area, as well as other areas of human-machine collaboration. See, however, this report's discussion of the "ilities".

4. Many leading AI researchers and scientists from other disciplines expressed their concerns of potential pitfalls of AI development in the "Open Letter on Artificial Intelligence." As the letter suggests, can we trust these agents to perform correctly? Can we verify and validate these agents with sufficient level of built-in security and control to ensure that these systems do what we want them to do?

JASON response: Verification and validation of AI agents is, at present, immature. There is considerable opportunity for DoD to participate in the program of advancing the state of the art of AI to become a true engineering discipline, in which V&V, as well as other engineering "ilities", will be appropriately controlled.

APPENDIX B: Briefings to JASON Study on Artificial Intelligence

JASON is grateful to the individuals listed below for both their formal presentations and several full days of wide-ranging discussions. While their views shaped the perspective of this report, these individuals are in no way responsible for JASON's findings and recommendations. We express thanks to sponsor David Han for his considerable commitment to the study in arranging these briefings.

June 27, 2016

Sponsor Brief-in: Stephen Welby, Assistant Secretary of Defense for Research and Engineering, "Autonomy / Artificial Intelligence Opportunities and Challenges"

Sponsor Brief-in: David Han, Associate Director for Basic Research in Machine Intelligence and Robotics, Office of the Assistant Secretary of Defense (R&E), "JASON Study on Artificial General Intelligence"

John Launchbury, Director I2O, DARPA, "A DARPA Perspective on Artificial Intelligence"

Tom Dietterich, President, Association for the Advancement of Artificial Intelligence and Distinguished Professor, Oregon State University, "Artificial Intelligence: Where We Are, Barriers to Progress, Where We Want to Be"

Subbarao Kambhampati, Arizona State University, "Planning Challenges in Human-Machine Collaboration"

Peter Norvig, Google, "Machine Learning and Artificial Intelligence"

Kenneth D. Forbus, Northwestern University, "Software Social Organisms: A Path to Human-Level AI"

June 28, 2016

Lynne E. Parker, Division Director, Information and Intelligent Systems Division, National Science Foundation, "NSF's Research Relevant to General AI"

Nicholas Roy, MIT Aero/Astro, "Representations vs. Algorithms: Robotics and AI"

Brian M. Sadler, Army Research Laboratory, "Distributed Collaborative Intelligent Systems: A Tactical Offset Strategy"

Marc Steinberg, Program Officer, Science of Autonomy, Office of Naval Research, "Artificial Intelligence, Autonomy, Machine Learning, and More..."

Steven 'Cap' Rogers, Senior Scientist (Autonomy), Air Force Research Laboratory, "The QuEST for Artificial General Intelligence"

John E. Laird, John L. Tishman Professor of Engineering, University of Michigan, "The Cognitive Architecture Approach to General Artificial Intelligence"

July 6, 2016

Honglak Lee, University of Michigan, "Deep Learning and General Intelligence"

Jitendra Malik, EECS, UC Berkeley, “Common Sense Computer Vision”

Martial Hebert, The Robotics Institute, Carnegie Mellon University, “Robust Perception and Reasoning”

Stuart Russell, Computer Science Division, UC Berkeley, “Prospects for General Intelligence”

APPENDIX C: The Back-Propagation Algorithm

Central to the DNN is its training, in which the millions of weights which connect the various neurons are assigned values. The modern technique for training these networks has its origins in the paper D. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature **323**(9), 533-536 (1986). Understanding this back-propagation algorithm is central to understanding why DNNs have benefited so greatly from big data and GPUs. It also helps to illuminate some of the challenges associated with making the nets work efficiently.

This Appendix summarizes the back-propagation algorithm. This derivation draws heavily from the book *Neural Networks and Deep Learning* (neuralnetworksanddeeplearning.com) by Michael Nielsen. It assumes some knowledge of multivariable calculus and linear algebra.

The material is organized into three sections. The first section, *Training a Single Neuron*, is where the formalism is developed. Here we see how data are used to train the net. In addition, stochastic gradient descent and the methodology for organizing the data is discussed.

The second section, *The Single Neuron, Multi-Layer Net*, uses the formalism detailed in the first section to explore how we put together multiple layers. In the end we learn why the gradient can be unstable in DNNs, and why that results in layers that train at different speeds.

In the final section, *The Multi-Neuron, Multi-Layer, Neural Net*, the matrix algebra approach to training large networks is derived. Organizing the calculation in terms of matrix algebra is what enables allows DNNs to benefit from GPUs.

C.1 Training a Single Neuron

Our simple model of the neuron, shown in Figure, is that the neuron has some inputs, x_i, where *i* runs from 1 to the number of inputs *I*. In what is to come, the inputs to neurons in layer *l* are the output of neurons in layer *l-1*. The output of a neuron is referred to as its activation, *a*. If we define our neuron as existing in layer 1, and the inputs are imagined to be from neurons in layer 0, then the inputs x_i can then be re-labeled as $a_i^0 = x_i$. In this notation superscripts index layers and subscripts index neurons.

The inputs to a neuron are combined as a linear sum, $z^1 = \sum_{i=1}^{I} w_i^1 a_i^0 + b$. The parameter *z* is called the weighted input to the neuron. The parameters w_i^1 are the weights and the parameter *b* is the bias. Note that the bias *b* can be thought of as the coefficient of the affine, and thus one can include the bias as a weight by writing $z = \sum_{i=0}^{I} w_i a_i^0$, where $w_0 = b$ and $a_0^j = 1$. When this formalism is used in multi-neuron layers in multi-layer DNNs, the weighted input to neuron *j* in layer *l* is given as $z_j^l = \sum_i w_{j,i}^l a_i^{l-1}$, where the sum over *i* includes all the neurons in layer *l*-1 and the affine.

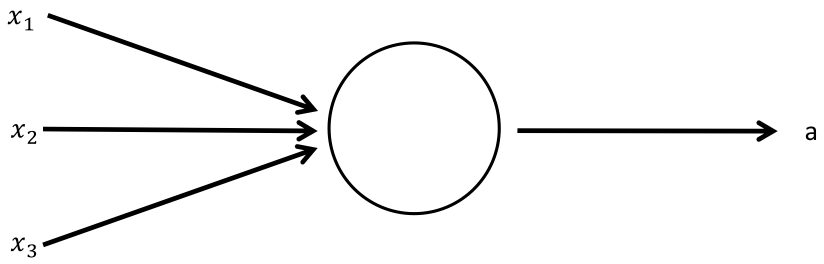

Figure C1: Schematic of the model neuron. The neuron has multiple inputs x_i and a single output. The output is called the activation a. The inputs to neurons in layer l are usually the activations of the neurons in the previous layer, $l-1$. The relation between the input and the output is described in the text.

The weighted input is acted on by the neuron, the output of which is a non-linear function, $a^1 = f(z^1)$. In our single neuron example, this is the final output of the system.

The only adjustable parameters in this system are the weights. Discovering optimal values of the weights is the process of training. This process generally requires training data, which are known input/output pairs (x, y). For example, if one were trying to train a DNN to approximate a function $y = f(x)$, the training data would be simply individual $(x, f(x))$ pairs, where *x* is a point in the domain of the function. Alternatively, if one were trying to train a DNN to recognize images, the input data *x* could be the intensity values of the pixels in the image, and the corresponding output *y* would be a description of the picture.

For the example of our single neuron, given an input vector x, the corresponding activation a^1 is calculated. The performance of the neuron is quantified using an error function, such as the quadratic function $E = \frac{1}{2}(y - a^1)^2$. The goal of training is to minimize the error function, which can be accomplished by techniques such as gradient descent. (W.H. Press, et. al., Numerical Recipes, Cambridge University Press, New York, 2007, Chapter 10.) The basic idea of gradient descent is that the weights are updated by using the gradient of the error function. For example, the component of the error function gradient for weight w is $\partial E/\partial w$. The weight is updated from w to $w + \delta w$, where $\delta w = -\eta\, \partial E/\partial w$. The parameter η is called the learning rate. Working through this for the weights in our single neuron example, one obtains

$$\delta w_i^1 = -\eta \frac{\partial E}{\partial w_i^1} = -\eta \frac{\partial E}{\partial a^1} \frac{\partial a^1}{\partial z^1} \frac{\partial z^1}{\partial w_i^1} = -\eta (a^1 - y)\, f'(z^1)\, a_i^0$$

Training starts by randomly assigning values to all the weights and biases in the network. Working from this initial state of the network, on then runs many input/output pairs through the network. For each input/output pair (x, y), the values for $\delta w_i^1(x, y)$ are calculated. These are then averaged over all *N* pairs of training data $\delta w_i^1 = \frac{1}{N}\sum_{(x,y)} \delta w_i^1(x, y)$. The weights are then updated, and the process begins again. In a perfect world, this continues until the weights find the global minimum of the error function averaged over all the training data, $E = \frac{1}{N}\sum_{(x,y)} E(x, y)$.

C.1.1 Summary

In addition to the formalism, there are several important ideas which can be addressed using this simple model

1) The training data are often vast, typically including many thousands to millions of (x, y) pairs. The complete set of input/output pairs used to train the network is typically randomly divided into three groups: a training set, a validation set, and a test set. Training data are used to train the network, as was described above. Validation data are used to measure how well the training is progressing and to adjust various parameters used in the training, such as the learning rate, the network architecture, etc. Additionally, you might evaluate the DNN against the validation data after each epoch as a benchmark for progress. When the training process is completed, the test data are used to measure the success of the final DNN.
2) Practical training of DNNs usually involves Stochastic Gradient Descent, in which the training data are randomly divided into mini-batches of size *M*. The point to this is that one can often obtain a reasonable approximation to the gradient by averaging over *M* samples, where $M \ll N$. This speeds up the learning process. The typical size of *M* is of order 10s of (x, y) pairs. The weights are updated by averaging the data in a single mini-batch. This is then repeated for all mini-batches until all of training data has been used. One complete run through all the training data is called an epoch. Training continues through many epochs until the DNN is sufficiently trained, typically involving several 10s of epochs.

C.2 The Single Neuron, Multi-Layer Net

Now consider the case of a single neuron, multi-layer network, such as that shown in Figure. This example serves only to develop formalism. Here there is a single input *x*, three neurons wired in series, and an output. The set of equations define the forward propagation of the input through the network:

Input Layer:

$$a_0^0 = 1, \qquad a_1^0 = \mathrm{x}$$

Layer 1:

$$z^1 = w_0^1 a_0^0 + w_1^1 a_1^0$$
$$a_0^1 = 1, \qquad a_1^1 = f(z^1)$$

Layer 2:

$$z^2 = w_0^2 a_0^1 + w_1^2 a_1^1$$
$$a_0^2 = 1, \qquad a_1^2 = f(z^2)$$

Layer 3:

$$z^3 = w_0^3 a_0^2 + w_1^3 a_1^2$$
$$a_1^3 = f(z^3)$$

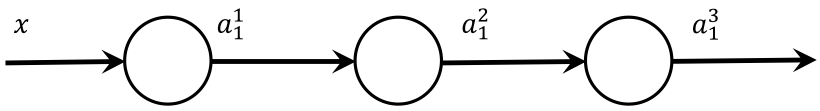

Figure C2: A network of three neurons in series, each with a single input. The activation of the i[th] neuron is a_1^i.

The output of the system is a_1^3, and this is what is evaluated in the error function

$$E = \frac{1}{2}(y - a_1^3)^2$$

Now we backwards propagate to calculate the derivatives necessary to adjust the weights. Let's start with the weights in the 3[rd] layer, w_0^3 and w_1^3, for which it is necessary to calculate

$$\delta w_i^3 = -\eta \frac{\partial E}{\partial w_i^3} = -\eta \frac{\partial E}{\partial z^3}\frac{\partial z^3}{\partial w_i^3}$$

The right-hand side contains the derivative $\epsilon^3 = \partial E/\partial z^3$, which is sufficiently easy to calculate

$$\epsilon^3 = \frac{\partial E}{\partial a_1^3}\frac{\partial a_1^3}{\partial z^3} = (a_1^3 - y)\, f'(z^3)$$

Additionally, the second factor is easy to generalize,

$$\frac{\partial z^3}{\partial w_i^3} = a_i^2$$

Thus, one obtains

$$\delta w_i^3 = -\eta \epsilon^3 a_i^2 = -\eta\, (a_1^3 - y)\, f'(z^3)\, a_i^2$$

Now let's calculate the update to the weights in the 2[nd] layer. We can formulate this exactly as was done above,

$$\delta w_i^2 = -\eta \frac{\partial E}{\partial w_i^2} = -\eta \frac{\partial E}{\partial z^2}\frac{\partial z^2}{\partial w_i^2} = -\eta \epsilon^2 a_i^1$$

The parameter ϵ^2 can be written in terms of ϵ^3,

$$\epsilon^2 = \frac{\partial E}{\partial z^3}\frac{\partial z^3}{\partial z^2} = \epsilon^3 \frac{\partial z^3}{\partial z^2}$$

Now we need to obtain $\partial z^3/\partial z^2$,

$$\frac{\partial z^3}{\partial z^2} = \frac{\partial (w_0^3 a_0^2 + w_1^3 a_1^2)}{\partial z^2} = w_0^3 \frac{\partial a_0^2}{\partial z^2} + w_1^3 \frac{\partial a_1^2}{\partial z^2}$$

But $\partial a_0^2/\partial z^2 = 0$ and $\partial a_1^2/\partial z^2 = f'(z^2)$, thus this expression simplifies to

$$\frac{\partial z^3}{\partial z^2} = w_1^3\, f'(z^2)$$

yielding

$$\epsilon^2 = \epsilon^3 w_1^3 f'(z^2)$$

The resulting updates to the weights are

$$\delta w_i^2 = -\eta \epsilon^2 a_i^2 = -\eta \epsilon^3 w_1^3 f'(z^2) a_i^1$$

Finally, lets update the weights in the first layer. The process is exactly as it was above, yielding the answer.

$$\delta w_i^1 = -\eta \epsilon^2 w_1^2 f'(z^1) a_i^0$$

In summary, the backwards propagation algorithm allows us to update the weights according to the following algorithm:

$$\delta w_i^3 = -\eta \, (a_1^3 - y) \, f'(z^3) \, a_i^2$$
$$\delta w_i^2 = -\eta \epsilon^3 w_1^3 f'(z^2) a_i^1$$
$$\delta w_i^1 = -\eta \epsilon^2 w_1^2 f'(z^1) a_i^0$$

where

$$\epsilon^l = \epsilon^{l+1} w_1^{l+1} f'(z^l)$$

C.2.1 Summary

We learn the following things about back-propagation

1) The update to weights in layer *l* are obtained from parameters that were calculated in the forwards propagation step, namely a_i^{l-1} and z^l, and a term related to layer l+1 which come from the backwards propagation step, namely ϵ^{l+1}. This is the beauty of the algorithm; you propagate forwards through the net to calculate the current values of the system, and then backwards propagate the error function through the net to update the weights.
2) If we write out long hand the explicit expression for the update to the weights, you obtain the following clear pattern:

$$\delta w_i^3 = -\eta \, (a_1^3 - y) \, f'(z^3) \, a_i^3$$
$$\delta w_i^2 = -\eta \, (a_1^3 - y)(\, f'(z^3) \, w_1^3) \, f'(z^2) \, a_i^2$$
$$\delta w_i^1 = -\eta \, (a_1^3 - y)(f'(z^3) \, w_1^3) \, (f'(z^2) \, w_1^2) \, f'(z^1) \, a_i^1$$

Thus, the update to the weights in the earlier layers involve ever more products of terms of the form $\left(f'(z^l) w_1^l\right)$. The function $f'(z)$ peaks at a value of 0.25. If the weights are distributed as a normal random variable, then it will usually be the case $\left|f'(z^l) w_1^l\right| < 1$. In this case the values δw_i^l will tend to get smaller as one works towards earlier layers. This is called the vanishing gradient problem. It results in the general rule of thumb that earlier layers tend to learn slower than later layers.

Of course, it does not have to be the case that $\left|f'(z^l) w_1^l\right| < 1$. It is possible to have large weights, $w_1^l \gg 1$, and biases that center the input weights z^i near to zero where $f'\left(z^i\right) \approx 0.25$, resulting in $w^i f'\left(z^i\right) > 1$. This tends to yield the opposite problem, a diverging gradient, in which earlier layers learn faster than later layers.

The underlying problem is not that the gradient is either vanishing or diverging, rather it is that the gradient is unstable. This comes from the fact that δw^l involves products from terms in all subsequent layers, and the product of many terms is unstable. As a result, different layers will learn at different rates, and the learning will be unbalanced. This problem becomes worse as the number of layers become large, and is thus a particular challenge for DNNs.

C.3 The Multi-Neuron, Multi-Layer, Neural Net

The formalism adopted above can be adapted to a fully connected, multi-neuron, multi-layer neural net. The addition of multiple neurons makes it natural to want to phase all the calculations in terms of matrix algebra. The derivation below starts with calculations of individual components, and then generalizes to the matrix formalism.

As was done above, the input data are defined to include the affine:

$$a_0^0 = 1, \qquad a_i^0 = \mathrm{x}_i \text{ for } i > 0$$

The description of the i^{th} neuron in layer l:

$$z_i^l = \sum_j w_{i,j}^l \, a_j^{l-1}$$

$$a_i^l = f(z_i^l)$$

It is straightforward to view all of these operations in terms of matrix algebra. In particular, if we format z^l as a $(N_l \text{ x } 1)$ column matrix, w^l as an $(N_l \text{ x } N_{l-1})$ rectangular matrix, and a^{l-1} as a $(N_{l-1} \text{ x } 1)$ column matrix, then the above expressions become

$$[z^l]_{(N_l \text{ x } 1)} = [w^l]_{(N_l \text{ x } N_{l-1})} \, [a^{l-1}]_{(N_{l-1} \text{ x } 1)}$$

$$[a^l]_{(N_l \text{ x } 1)} = [f(z^l)]_{(N_l \text{ x } 1)}$$

The error function is then

$$E = \frac{1}{2} [y - a^L]^T_{(1 \text{ x } N_l)} \, [y - a^L]_{(N_l \text{ x } 1)}$$

where the notation $[\quad]^T$ is meant to mean the transpose matrix.

Now start the backwards propagation exercise. The update to the weights in the final layer is

$$\delta w_{i,j}^L = -\eta \frac{\partial E}{\partial w_{i,j}^L} = -\eta \frac{\partial E}{\partial z_i^L} \frac{\partial z_i^L}{\partial w_{ij}^L} = -\eta \epsilon_i^L a_j^{L-1}$$

where the derivative $\epsilon_i^L = \partial E / \partial z_i^L$. It is straightforward to calculate

$$\epsilon_i^L = \frac{\partial E}{\partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L} = (a_i^L - y_i) \, f'(z_i^L)$$

Thus, one obtains

$$\delta w_{i,j}^L = -\eta \, (a_i^L - y_i) \, f'(z_i^L) \, a_j^{L-1}$$

Now lets calculate the update to the weights in the l^{th} layer. We can formulate this exactly as was done above,

$$\delta w_{i,j}^{l} = -\eta \frac{\partial E}{\partial w_{i,j}^{l}} = -\eta \frac{\partial E}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{i,j}^{l}} = -\eta \epsilon_i^{l} a_j^{l-1}$$

The parameter ϵ_i^{l} can be written in terms of ϵ_j^{l+1},

$$\epsilon_i^{l} = \sum_j \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^{l}} = \sum_j \epsilon_j^{l+1} w_{j,i}^{l+1} f'\left(z_i^{l}\right)$$

where the sum is over the N_{l+1} neurons in the $(l+1)^{th}$ layer. Now, the interesting part of this expression is that it can be re-cast in terms of matrices. In particular, if we format ϵ^{l} as a $(N_l \text{ x } 1)$ column matrix, w^{l+1} as an $(N_{l+1} \text{x } N_l)$ matrix, and $f'(z^{l})$ as a $(N_l \text{ x } N_l)$ diagonal matrix with the terms $f'\left(z_i^{l}\right)$ along the diagonal, then the expression for ϵ^{l} becomes

$$[\epsilon^{l}]_{(N_l \text{ x } 1)} = [f'(z^{l})]_{(N_l \text{ x } N_l)} \, [w^{l+1}]^{T}_{(N_l \text{ x } N_{l+1})} \, [\epsilon^{l+1}]_{(N_{l+1} \text{ x } 1)}$$

Using the same notation, one can express the update to the weights as

$$[\delta w^{l}]_{(N_l \text{ x } N_{l-1})} = -\eta \, [\epsilon^{l}]_{(N_l \text{ x } 1)} [\, a^{l-1}]^{T}_{(1 \, x N_{l-1})}$$

And thus backwards propagation enables the calculation of the update to the weights in a particular layer in terms of parameters calculated during forward propagation and in terms of parameters calculated for subsequent layers during the backwards propagation.

Similarly, we can re-cast the backwards propagation results for the final layer L using this same formalism,

$$[\epsilon^{L}]_{(N_L \text{ x } 1)} = [f'(z^{L})]_{(N_L \text{ x } N_L)} \, [a^{L} - y]_{(N_L \text{ x } 1)}$$

$$[\delta w^{L}]_{(N_L \text{ x } N_{L-1})} = -\eta \, [\epsilon^{l}]_{(N_L \text{ x } 1)} [\, a^{L-1}]^{T}_{(1 \, x N_{L-1})}$$

C.3.1 Summary

The entire training algorithm can be written in matrix notation. The forward propagation is written as

$$[z^{l}] = [w^{l}] \, [a^{l-1}]$$

$$[a^{l}] = [f(z^{l})]$$

The backwards propagation starts with the parameters for the final layer

$$[\epsilon^{L}] = [f'(z^{L})] \, [a^{L} - y]$$

$$[\delta w^{L}] = -\eta \, [\epsilon^{L}] \, [\, a^{L-1}]^{T}$$

The remaining layers are calculated using the following relations.

$$[\epsilon^{l}] = [f'(z^{l})] \, [w^{l+1}]^{T} \, [\epsilon^{l+1}]$$

$$[\delta w^{l}] = -\eta \, [\epsilon^{l}] \, [\, a^{l-1}]^{T}$$

This matrix algebra formalism is ideally suited to benefit from the parallelism of graphics processor units (GPUs). Graphics processors were initially developed for processing image data for displays, which are also dominated by matrix operations. The benefits become all the more important as the scale of the problem grows, both in terms of the number of neurons per layer and the total number of layers. Thus, the success of deep neural networks relies on availability of GPU hardware.

APPENDIX D: List of Acronyms Used

ACTUV	Anti-Submarine Warfare Continuous Trail Unmanned Vessel
AGI	Artificial General Intelligence
AI	Artificial Intelligence
AI100	One Hundred Year Study on Artificial Intelligence
ASD (R&E)	Assistant Secretary of Defense for Research and Engineering
AVX	Advanced Vector Extension
BBC	British Broadcasting Corporation
BD	Big Data
BD/DL	Big Data / Deep Learning
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CTBTO	Comprehensive Nuclear Test Ban Treaty Organization
DL	Deep Learning
DNN	Deep Neural Network
DOD	Department of Defense
DRAM	Dynamic Random Access Memory
eDRAM	Embedded DRAM
EIE	Efficient Inference Engine
FLOPS	Floating-point Operations Per Second
GAN	Generalized Adversarial Network
GOP	Giga Operations
GPU	Graphics Processor Unit
GRU	Gated Recurrent Unit (element in an RNN)
HMM	Hidden Markov Model
KRR	Knowledge Representation and Reasoning
LSTM	Long and Short-Term Memory (element in an RNN)
MAC	Memory Access
MKL	Math Kernel Library
ML	Machine Learning

MOSIS	Metal Oxide Semiconductor Implementation Service
NLP	Natural Language Processing
NN	Neural Network
OSD	Office of the Secretary of Defense
PGM	Probabilistic Graphical Model
PPL	Probabilistic Programming Language
RAM	Random Access Memory
ReLU	Rectified Linear Unit (nonlinear element in NN)
RGB	Red, Green, Blue
RL	Reinforcement Learning
RNN	Recursive Neural Network
ROI	Region Of Interest
SAT	Boolean Satisfiability Problem
SIMD	Single Instruction, Multiple Data
SRAM	Static Random Access Memory
TF	Tera FLOPS
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
UCAS	Unmanned Combat Air System
VGG	Visual Geometry Group (Oxford University)
VPU	Vision Processing Unit