

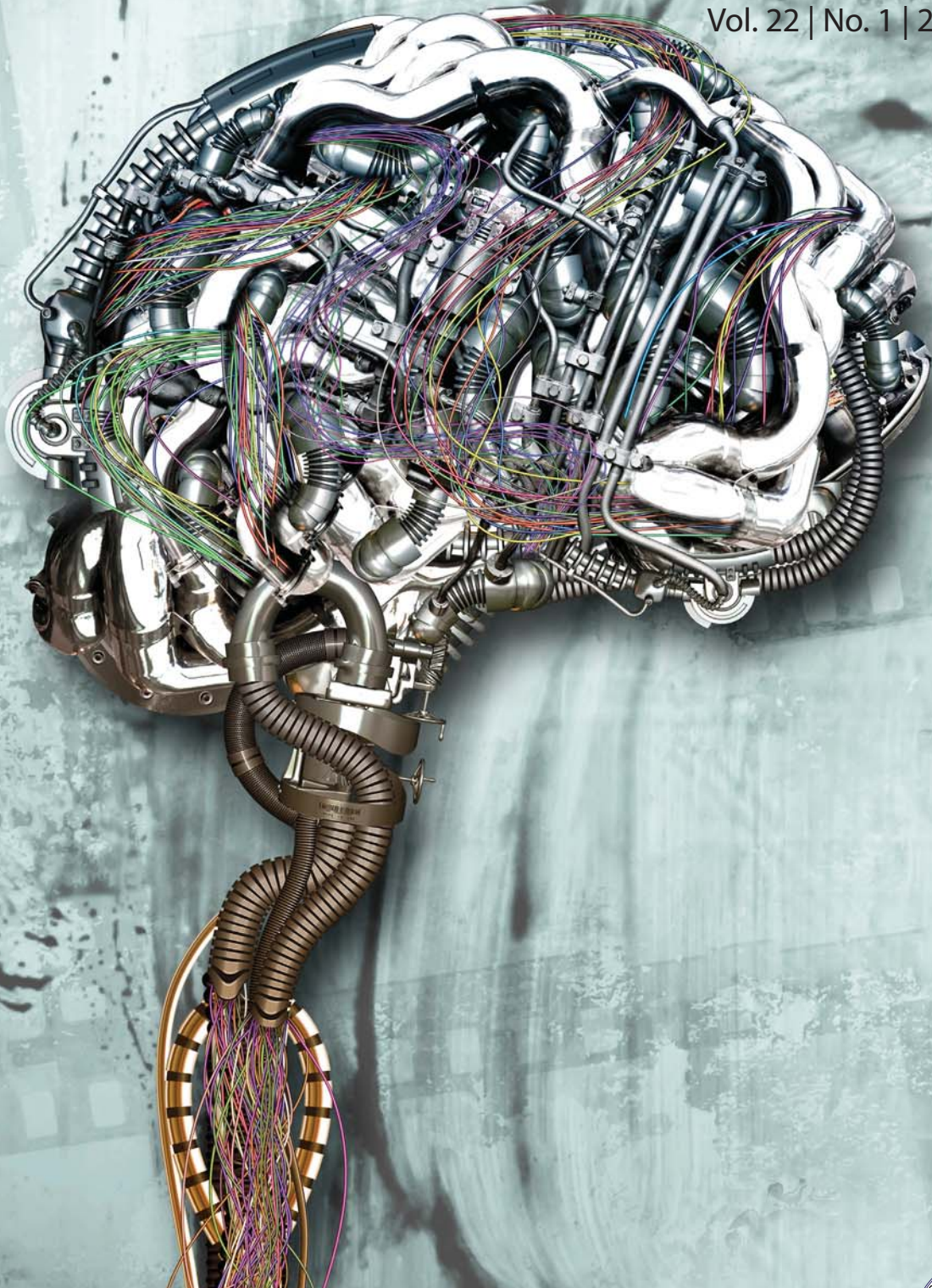


THE

Next Wave

The National Security Agency's review of emerging technologies

Vol. 22 | No. 1 | 2018



Machine Learning

[Photo credit: diuno/iStock/Thinkstock]



GUEST **Editor's column**

Joe McCloskey & David J. Mountain

Machine learning—often described as artificial intelligence or deep learning—is in the news today, almost everywhere. *The MIT Technology Review* did a special issue on the topic for their November/December 2017 issue. *The New York Times* has done a series of articles on the topic, most recently in a *New York Times Magazine* article published November 21, 2017, titled “Can A.I. be taught to explain itself.”

Additionally, it was the focus of a special panel discussion at the November 2017 IEEE International Conference on Rebooting Computing titled “AI, cognitive information processing, and rebooting computing: When will the new sheriff come to tame the wild, wild, west?” As this last item suggests, there can be considerable difficulty in separating fact from fiction, and reality from hype, in this technical field.

In this, the first of two special issues of *The Next Wave* on machine learning, we hope to provide you with a reasoned look at this topic, highlighting the various facets of NSA research and collaborations in the field.

In our first article, NSA researcher Steve Knox gives “Some basic ideas and vocabulary in machine learning.”

“Machine learning for autonomous cyber defense,” by NSA researcher Ahmad Ridley provides insight into a strategically valuable mission application of these techniques.

NSA researchers Mark Mclean and Christopher Krieger focus our attention on how these techniques might alter computing systems in “Rethinking neuromorphic computing: Why a new paradigm is needed.”

In “How deep learning changed computer vision,” NSA researchers Bridget Kennedy and Brad Skaggs help us understand the particular mechanisms that made these techniques so ubiquitous.

“Deep learning for scientific discovery,” by Pacific Northwest National Laboratories researchers Courtney Corley et al. present several topic areas in which modern representation learning is driving innovation.

We finish this issue with an article noting some pitfalls in the use of machine learning, “Extremely rare phenomena and sawtooths in La Jolla,” by researchers at the Institute for Defense Analyses’ Center for Communications Research in

Contents

2 Some Basic Ideas and Vocabulary in Machine Learning

STEVEN KNOX

7 Machine Learning for Autonomous Cyber Defense

AHMAD RIDLEY

15 Rethinking Neuromorphic Computation: Why a New Paradigm is Needed

MARK R. MCLEAN

CHRISTOPHER D. KRIEGER

23 How Deep Learning Changed Computer Vision

BRIDGET KENNEDY

BRAD SKAGGS

27 Deep Learning for Scientific Discovery

COURTNEY CORLEY

NATHAN HODAS, ET AL.

32 Extremely Rare Phenomena and Sawtooths in La Jolla

ANTHONY GAMST

SKIP GARIBALDI

38 AT A GLANCE: Machine Learning Programs Across the Government

41 FROM LAB TO MARKET: AlgorithmHub Provides Robust Environment for NSA Machine Learning Research



La Jolla, California, Anthony Gamst and Skip Garibaldi.

Our intent with this special issue is to enhance your understanding of this important field and to present the richness of our research from a variety of perspectives. We hope you enjoy our perspective on the topic.

Joe McCloskey

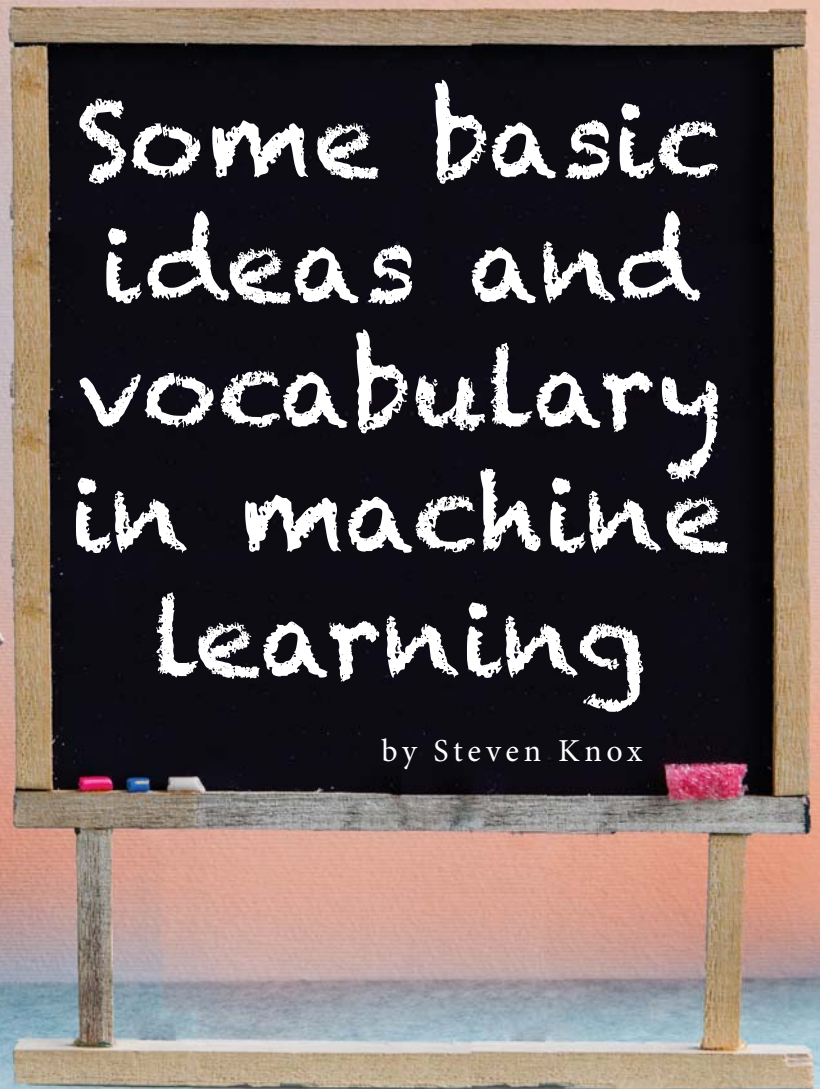
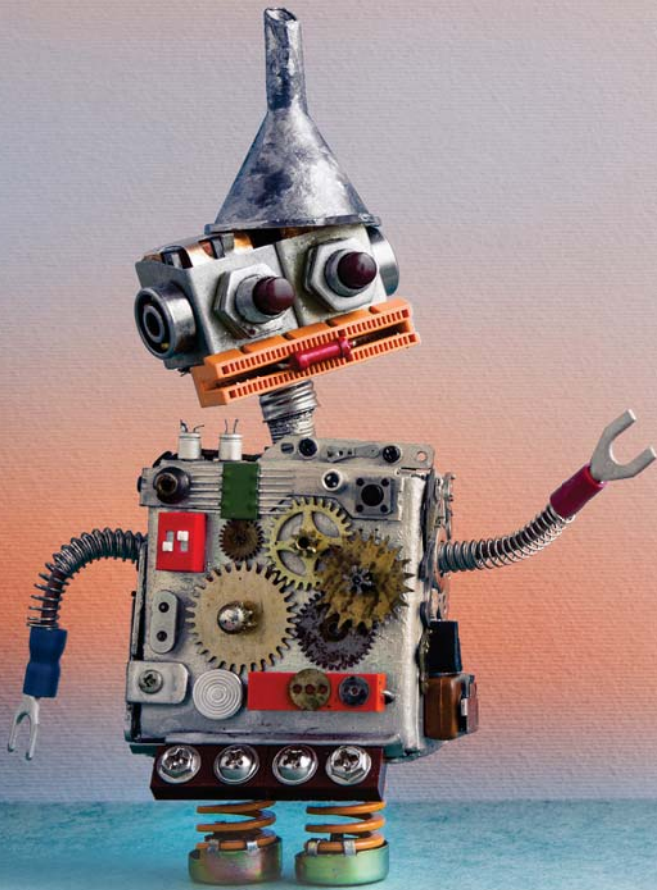
Deputy Technical Director
Research Directorate, NSA

David J. Mountain

Advanced Computing Systems
Research Program
Research Directorate, NSA

The Next Wave is published to disseminate technical advancements and research activities in telecommunications and information technologies. Mentions of company names or commercial products do not imply endorsement by the US Government. The views and opinions expressed herein are those of the authors and do not necessarily reflect those of the NSA/CSS.

This publication is available online at <http://www.nsa.gov/thenextwave>. For more information, please contact us at TNW@tycho.ncsc.mil.



[Photo credit: Besjunior/iStock/Thinkstock]

What is Machine Learning? *Machine learning (ML)* is a difficult, perhaps impossible, term to define without controversy. It is tempting to avoid defining it explicitly and just let you, the reader, learn a definition inductively by reading, among other sources, this issue of *The Next Wave*.¹ You could then join the rest of us in any number of arguments about what ML is and what ML is not.

For the present purpose, a *machine* is an artificial device (hardware, software, or an abstraction) which takes in an input and produces an output. Phrased another way, a machine takes in a stimulus and produces a response. *Machine learning (ML)* refers to a process whereby the relationship between input and

output (or stimulus and response) changes as a result of experience. This experience may come in the form of a set of stimuli paired with corresponding desired responses, or in the form of sequential stimuli accompanied by rewards or punishments corresponding to the machine's responses. This definition, while perhaps not entirely satisfying, has at least the advantages of being very general and being in accordance with early references [1, 2].

In this introduction we shall think of a machine as a mathematical abstraction, that is, simply as a *function* which maps inputs to outputs. This function may be deterministic, or it may be in some respect random (i.e., two identical inputs may produce different

1. We avoid circular reasoning here by assuming the reader is not a machine.

outputs). Working at this level of abstraction enables us to use precise mathematical language, to apply useful, thought-clarifying ideas from probability, statistics, and decision theory, and most importantly, to address with generality what is common across a wide variety of applications.² In keeping with this abstraction, the goal of ML is to solve

The Problem of Learning. There are a known set X and an unknown function f on X . Given data, construct a good approximation \hat{f} of f . This is called learning f .

There is a lot packed into this simple statement. Unpacking it will introduce some of the key ideas and vocabulary used in ML.

Key Ideas. The domain of the unknown function f , the set X , is called *feature space*. An element $X \in X$ is called a *feature vector* (or an *input*) and the coordinates of X are called *features*. Individual features may take values in a continuum, a discrete, ordered set, or a discrete, unordered set. For example, an email spam filter might consider features such as the sender's internet protocol (IP) address (an element of a finite set), the latitude and longitude associated with that IP address by an online geolocation service (a pair of real numbers), and the sender's domain name (a text string). Features may arise naturally as the output of sensors (for example, weight, temperature, or chemical composition of a physical object) or they may be thoughtfully engineered from unstructured data (for example, the proportion of words in a document which are associated with the topic "machine learning").

The range of the unknown function f , the set $f(X)$, is usually either a finite, unordered set or it is a continuum. In the first case, learning f is called *classification* and an element $Y \in f(X)$ is called a *class*. In the second case, learning f is called *regression* and an element $Y \in f(X)$ is called a *response*. The bifurcation of terminology based on the range of f reflects different historical origins of techniques for solving the problem of learning in these two cases.

Terminology also bifurcates based on the nature of the observed data, in particular, whether or not the range of f is observed directly. If the observed data have the form of matched stimulus-response pairs,

$$(x_1, y_1), \dots, (x_n, y_n) \in X \times f(X),$$

then the data are called *marked data* and learning f is called *supervised learning* ("supervised" because we observe inputs x_1, \dots, x_n to the unknown function f and also the corresponding, perhaps noisy, outputs y_1, \dots, y_n of f). If the observed data have the form

$$x_1, \dots, x_n \in X,$$

then the data are called *unmarked data* and learning f is called *unsupervised learning*. While unsupervised learning may, on its face, sound absurd (we are asked to learn an unknown function based only on what is put into it), two cases have great utility: when the range of f is discrete, in which case unsupervised learning of f is *clustering*; and when x_1, \dots, x_n are a random sample from a probability distribution with density function f , in which case unsupervised learning of f is *density estimation*. A situation in which both marked and unmarked data are available is called *semi-supervised learning*.

We generally consider data to be random draws (X, Y) from some unknown joint probability distribution³ $P(X, Y)$ on $X \times f(X)$. This is true even if the outputs of f are unobserved, in which case we can think of the unobserved Y 's as *latent variables*. This is not to say that we believe the data generation process is intrinsically random—rather, we are introducing probability $P(X, Y)$ as a useful descriptive language for a process we do not fully understand.

An approximation \hat{f} of f could be considered "good" if, on average, the negative real-world consequences of using \hat{f} in place of f are tolerable. In contrast, an approximation \hat{f} of f could be considered "bad" if the negative real-world consequences are intolerable, either through the accumulation of many small errors or the rare occurrence of a disastrous error. The consequences of errors are formalized and quantified

2. Of course, an ML solution for any given application must be realized in actual hardware and software, or the whole exercise is pointless. It is hard to overstate the importance of good hardware and software.

3. The joint distribution $P(X, Y)$ can be factored into the conditional distribution of Y given X , $P(Y | X)$, times the marginal distribution of X , $P(X)$, which is the distribution of X once Y has been "averaged away". That is, $P(X, Y) = P(Y | X)P(X)$. It can also be factored the other way, $P(X, Y) = P(X | Y)P(Y)$. Both factorizations are useful and inform the design of ML algorithms.

by a *loss function*, $L(y, \hat{f}(x))$. At a point x in feature space \mathcal{X} , $L(y, \hat{f}(x))$ compares the true class or response $y = f(x)$ to the approximation $\hat{f}(x)$ and assigns a non-negative real number to the cost or penalty incurred by using $\hat{f}(x)$ in place of y . Commonly used loss functions in regression are *squared-error loss*, where $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$, and *absolute-error loss*, where $L(y, \hat{f}(x)) = |y - \hat{f}(x)|$. A commonly used⁴ loss function in classification is *zero-one loss*, where $L(y, \hat{f}(x)) = 0$ if $y = \hat{f}(x)$ and $L(y, \hat{f}(x)) = 1$ if $y \neq \hat{f}(x)$. In classification with C classes; that is, when the range $f(\mathcal{X})$ can be identified with the set of labels $\{1, \dots, C\}$, an arbitrary loss function can be specified by the cells of a $C \times C$ *loss matrix* L . The loss matrix usually has all-zero diagonal elements and positive off-diagonal elements, where $L(c, d)$ is the loss incurred for predicting class d when the true class is c for all $1 \leq c, d \leq C$ (so correct classification incurs no loss, and misclassification incurs positive loss).⁵

Some techniques, such as neural networks, solve a C -class classification problem by estimating a conditional probability distribution on classes $1, \dots, C$, given a feature vector X ,

$$(\hat{P}(Y = 1 | X), \dots, \hat{P}(Y = C | X)).$$

In these techniques, which essentially translate a classification problem into a regression problem, an observed class label y is identified with a degenerate probability distribution on the set of classes $\{1, \dots, C\}$. Called a *one-hot encoding* of y , this is a C -long vector which has the value 1 in position y and 0 elsewhere. A loss function commonly used to compare an estimated probability distribution to a one-hot encoding of a class label is *cross-entropy loss*, $L(y, (\hat{P}(Y=1 | X), \dots, \hat{P}(Y=C | X))) = -\log \hat{P}(Y=y | X)$, although squared-error loss is also used sometimes in this setting.

The choice of loss function is subjective and problem dependent. Indeed, it is the single most important

control we have over the behavior of ML algorithms⁶ because it allows us to encode into the algorithm our values with respect to the real-world application: specifically, by stating the cost of each possible type of error the algorithm could make. That said, loss functions are often chosen for convenience and computational tractability.⁷ When approximations \hat{f} are obtained by fitting statistical models, squared-error loss and cross-entropy loss can be derived in certain situations from the (subjective) principle that the best approximation \hat{f} in a class of models is one which assigns maximal likelihood to the observed data.

To rephrase, then, an approximation \hat{f} of f could be considered “good” if its use incurs small loss *on average* as it is applied to data drawn from the joint probability distribution $P(X, Y)$. The average loss incurred by approximation \hat{f} is its *risk*. There are at least three different types of risk, depending on what is considered fixed and what can vary in future applications of \hat{f} . The *risk of approximation \hat{f} at point $x \in \mathcal{X}$* is the expected loss incurred by using $\hat{f}(x)$ in place of Y for new data (X, Y) such that $X = x$, $E_{Y|X=x}[L(Y, \hat{f}(x))]$.⁸ The response Y is treated as random, with the conditional distribution $P(Y | X = x)$, while the input $X = x$ and trained classifier \hat{f} are treated as fixed. Choosing an approximation \hat{f} to minimize the risk of \hat{f} at a specific point (or finite set of points) in \mathcal{X} is called *transductive learning*. It is of use when the entire set of points at which predictions are to be made is known at the time that \hat{f} is constructed. The *risk of approximation \hat{f}* is the expected loss incurred by using $\hat{f}(X)$ in place of Y for new data (X, Y) , $E_{X,Y}[L(Y, \hat{f}(X))]$. Data point (X, Y) is treated as random while the trained classifier \hat{f} is treated as fixed. Choosing an approximation \hat{f} to minimize the risk of \hat{f} is done when the points at which \hat{f} is to make a prediction are not known at the time that \hat{f} is constructed, but will be drawn from the marginal distribution $P(X)$ on \mathcal{X} at some future time. This is the typical situation in applied ML.

4. Zero-one loss is commonly used in textbook examples. It is rarely, if ever, appropriate in applications.

5. In the spam-filtering example, “loss” is measured in inconvenience to the email recipient. If we define the inconvenience of reading a spam email as one unit of loss, then the loss incurred by labeling a spam email as not spam is $L(\text{spam}, \text{not spam}) = 1$. Relative to this, different recipients may have different opinions about the inconvenience of having a non-spam email labeled as spam. That is, $L(\text{not spam}, \text{spam}) = c$, where each recipient has his or her own value of $c > 0$: some users might have $c = 10$, some $c = 20$, and perhaps some might even have $c < 1$.

6. In this statement, we are ignoring for the moment the critically important issue of optimization required for training most ML algorithms. Another control available in some, but not all, ML methods, is the marginal (or prior) distribution on classes, $(P(Y = 1), \dots, P(Y = C))$.

An *approximation method* is a function which maps training data sets to approximations of f . It maps a set of n observed data $S = ((x_1, y_1), \dots, (x_n, y_n)) \in (\mathcal{X} \times f(\mathcal{X}))^n$ to a function \hat{f}_S , where \hat{f}_S is a function that maps $\mathcal{X} \rightarrow f(\mathcal{X})$. Actually, approximations \hat{f}_S lie in a method-specific proper subset of the set of functions $\mathcal{X} \rightarrow f(\mathcal{X})$, about which more will be said later. The probability distribution on $\mathcal{X} \times f(\mathcal{X})$ extends to training data sets S and thence to approximations \hat{f}_S , so approximations \hat{f}_S are random variables. The *risk of an approximation method (trained on data sets of size n)* is the expected loss incurred by drawing a random training set S of size n , constructing an approximation \hat{f}_S from it, and using $\hat{f}_S(X)$ in place of Y for new data (X, Y) , $E_{S, X, Y}[L(Y, \hat{f}_S(X))]$. Choosing an approximation method $S \rightarrow \hat{f}_S$ to minimize the risk of the approximation method is done by algorithm designers, whose goal is to produce approximation methods that are useful in a wide variety of as-yet-unseen applied problems. Note that there is some tension between the goal of the algorithm designer, who wants to obtain low-risk approximations \hat{f}_S from most data sets S , and the goal of the applied ML practitioner, who wants to obtain a low-risk approximation \hat{f} from the single data set he or she has to work with in a given application.

Risk estimation, model training, and model selection

Since the joint probability distribution $P(X, Y)$ is unknown—it is, after all, simply a name for an unknown process which generates the data we observe—the actual risk of approximation \hat{f} , in any of the three senses above, cannot be computed. Solving an ML problem in practice means searching for an approximation \hat{f} that is optimal according to a computable optimality criterion, which is an approximation of, or surrogate for, the true risk. The risk of \hat{f} can be estimated in multiple ways, each of which has advantages and

disadvantages. There is the *training estimate of risk*, $\frac{1}{n} \sum_{i=1}^n L(y_i, \hat{f}(x_i))$, where $(x_1, y_1), \dots, (x_n, y_n)$ are the data used to produce approximation \hat{f} ; there is the *validation (or test) estimate of risk*, $\frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} L(\tilde{y}_i, \hat{f}(\tilde{x}_i))$, where $(\tilde{x}_1, \tilde{y}_1), \dots, (\tilde{x}_{\tilde{n}}, \tilde{y}_{\tilde{n}})$ are *hold-out data* which are not used to produce approximation \hat{f} ; and there is the *k -fold cross-validation estimate of risk*, $\frac{1}{k} \sum_{i=1}^k \frac{1}{|S_k|} \sum_{(x,y) \in S_k} L(y, \hat{f}_k(x))$, where $S_1 \cup \dots \cup S_k$ is a partition of the set $\{(x_1, y_1), \dots, (x_n, y_n)\}$ into k subsets and approximation \hat{f}_k is produced using all of the data except those in S_k , which is treated as hold-out data for \hat{f}_k . In addition to computable estimates of risk, other computable optimality criteria include surrogates derived from asymptotic behavior as $n \rightarrow \infty$, such as Akaike’s Information Criterion [3], and computable upper bounds for the true risk.

Given a computable optimality criterion, the practical problem of finding an approximation \hat{f} which is exactly or approximately optimal with respect to that criterion must be solved. Each concrete method for solving the problem of learning specifies (explicitly or implicitly) a set \mathcal{F} of functions which map $\mathcal{X} \rightarrow f(\mathcal{X})$ and provides a method for searching the set \mathcal{F} for an optimal member. For example, in classical linear regression, $\mathcal{X} = \mathbb{R}^m$ for some $m \geq 1$ and the set \mathcal{F} is the set of all affine functions $\mathbb{R}^m \rightarrow \mathbb{R}$; that is, all functions of the form $f(x_1, \dots, x_m) = \beta_0 + \beta_1 x_1 + \dots + \beta_m x_m$. Using squared-error loss and using the training estimate of risk, the computable optimality criterion is $\frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_{i,1} - \dots - \beta_m x_{i,m})^2$. Finding the optimal approximation $f \in \mathcal{F}$, which is equivalent to finding optimal values of the parameters β_0, \dots, β_m , can be done approximately by using an iterative gradient descent method (the optimal \hat{f} actually has a closed-form solution in this case, but closed-form solutions are not typical in ML).

Specification of a ML method, and hence specifying the set of functions \mathcal{F} to be searched, is called *model*

7. Many ML methods require solving optimization problems through iterative numerical procedures because no closed-form solutions are known. It has been found empirically that some loss functions lead to easier optimization problems than others, and that sometimes changing a loss function to make an optimization problem easier, say by replacing cross-entropy loss with squared-error loss in a neural network, results in a better solution, even in terms of the original loss function.

8. Recall that the expected value is an integral or sum with respect to the appropriate probability distribution. If Y is real-valued (regression), $E_{Y|X=x}[L(Y, \hat{f}(x))] = \int_{-\infty}^{\infty} L(y, \hat{f}(x)) dP(y|X=x)$, while if Y takes values in the discrete set $\{1, \dots, C\}$ (classification), $E_{Y|X=x}[L(Y, \hat{f}(x))] = \sum_{y=1}^C L(y, \hat{f}(x)) P(Y=y|X=x)$.

selection. Searching a set \mathcal{F} for an approximately optimal member is called *training a model*. If the functions in \mathcal{F} are all described by a finite number of parameters which does not depend on the number of data, n , then the method is called *parametric*, and otherwise it is called *nonparametric*. An important concern in model selection is to specify a set of functions \mathcal{F} such that the optimal member \hat{f} is sufficiently well adapted to the training data (i.e., it is not *underfit* to the data) but at the same time is not too well adapted, essentially memorizing the correct outputs to the given inputs but unable to generalize (i.e., not *overfit* to the data). Avoidance of overfitting can be done by *regularization*, which essentially means modifying the loss function to penalize model complexity in addition to penalizing model inaccuracy.

Machine learning and statistics


From the way it has been presented in this introduction, ML appears very statistical. One might ask whether there is, in fact, any meaningful difference between ML and statistics. An answer to this is that there is considerable overlap in methodology (indeed, many ML techniques are based upon statistical models), but the applied focus tends to be different: Statistics

is more often about producing human insight from data about an incompletely understood process (the unknown function f), while ML is more often about producing an automatic means of decision-making which, for a variety of reasons,⁹ might be preferred to human decision making [4]. On some level, debating whether a particular model or method is statistics or ML is like debating whether a sledgehammer is a construction tool or a demolition tool—the answer depends on what one hits with it, and why. 📌

References

- [1] Hartree DR. *Calculating Instruments and Machines*. Urbana (IL): University of Illinois Press; 1949. p. 70.
- [2] Turing AM. “Computing machinery and intelligence.” *Mind*. 1950;59(236):454–460.
- [3] Akaike H. “Information theory and an extension of the maximum likelihood principle.” In: Kotz S, Johnson NL (editors). *Breakthroughs in Statistics*. Springer Series in Statistics (Perspectives in Statistics). New York (NY): Springer; 1992. p. 610–624.
- [4] Breiman L. “Statistical modeling: The two cultures.” *Statistical Science*. 2001;16(3):199–231.

9. Consistency, reliability, scalability, speed, deployment in specific environments, etc.



Machine learning for autonomous cyber defense

by Ahmad Ridley

Imagine networks where zero day cannot happen to anybody, where zero day does not guarantee a hacker's success, where defenders work together with guardian machines to keep networks safe. Imagine getting a text message from the system that protects your business, letting you know it just learned about a new flaw in your document reader and synthesized a new patch all on its own.

Mike Walker, Defense Advanced Research Projects Agency (DARPA) Program Manager,
Remarks at opening of the DARPA Grand Cyber Challenge, August 4, 2016

The number and impact of cyberattacks continue to increase every year. The NSA Research Directorate aims to build an autonomous cyber defense system that will make enterprise networks, and their associated missions and services, more resilient to cyberattacks. Such a system should choose appropriate responses to help these networks achieve their mission goals and simultaneously withstand, anticipate, recover, and/or evolve in the presence of cyberattacks. Through automation, the system should make decisions and implement responses in real time and at network scale. Through adaptation, the system should also learn to reason about the best response based on the dynamics of both constantly evolving cyberattacks and network usage and configurations.

In this article, we present our research methodology and investigate a branch of machine learning (ML) called reinforcement learning (RL), to construct an autonomous cyber-defense system that will control and defend an enterprise network.

Cyber-resilience research methodology

We seek to change the enterprise network defense paradigm by integrating autonomous (i.e., automated and adaptive) cyber defense within the defensive capabilities of large and complex mission network systems. We define *cyber resilience* as the ability of missions and services to maintain a certain level of performance despite the presence of sophisticated cyber adversaries in the underlying enterprise network [1]. Cyber-defense strategies that engage potential adversaries earlier in the cyber kill chain are fundamental to cyber resilience. The cyber kill chain is a model of the various stages of a cyberattack, listed in increasing order of severity for the defender. Early adversary engagement allows the defender to increase detection confidence and disambiguate malicious from nonmalicious actors. Ultimately, we must develop a scientific method to measure cyber-resiliency and understand the effects of cyber responses on those measurements. Such development is critical to the evolution of our research.

Cyber-resiliency research presents a set of challenges, detailed in table 1 below, including:

1. Detecting the presence and movement of a sophisticated cyber attacker in a network,
2. Automating the speed at which human analysts can react to a large volume of cyber alerts,
3. Mitigating the imbalance in workload between defender and attacker, and
4. Allowing for the fact that autonomous systems are still in the research phase across many disciplines.

Autonomous cyber defense

Our research focuses on reasoning and learning methods to enable automated and self-adaptive decisions and responses. A cyber-defense system based on this research is autonomous, which will allow it to effectively improve the cyber resilience of a mission or service. Once the system receives data, analytic results, and state representation of the network, its decision engine must reason over the current state knowledge, attacker tactics, techniques, and procedures (TTPs), potential responses, and desired state. The system will determine a response strategy to best achieve the goals

TABLE 1. Cyber-Resiliency Challenges: Fundamental assumptions, principles, and hard problems

Fundamental Assumption	Guiding Principle	Hard Problem
Adversary can always get inside the system but cannot hide from all vantage points	Utilize diverse sensors from across the environment to form a holistic and contextual understanding of adversary activities	Dynamic information collection and control of diverse sensors
		Real-time fusion and contextualization of events from diverse and disparate data sources
Humans cannot quickly identify and react to adversarial behavior	Automated reasoning and adaptive response mechanisms are needed to mitigate damage from attacks at adversarial speed and enterprise scale	Representing and reasoning over system state and response effects on systems and adversaries
	Apply automated responses to increase certainty and situational understanding	Automated reasoning and response in the face of uncertain and missing data
Cyber defenders have asymmetric disadvantage	Defensive deception and adaptive response can improve the asymmetric imbalance	Developing a common language for automated orchestration of responses
		Determining how best to use deception—where, when, and what kind
Autonomous systems are still in research phase across many disciplines	Adaptive cyber-defense systems must allow for operation with varied levels of human interaction	Create metrics and experiments to evaluate the effectiveness and impact on attacker behavior based on principles from human science
		Establishing human trust in automated cyber systems
		Developing secure design methodology for adaptive cyber-defense systems

of the mission or service, which rely on the underlying enterprise network. This reasoning must account for incomplete and untrustworthy data from host and network sensors. To make sequential decisions in the presence of this uncertainty, the system should receive full-loop feedback about its decisions and responses by selecting and taking responses, observing the effects, and learning from these observations to improve performance over time.

To maximize the benefits of full-loop reasoning and response, sensors collect observable data, such as host logins or network connections between machines. The data is analyzed to extract key system features and conditions, such as unexpected internet connections or an unusual number of failed authentications. These features and conditions are combined to create a system state. Decisions are made based on the quality of this state, such as the frequency or severity of unwanted conditions, and responses are employed to optimize the quality of state subject to the cybersecurity and/or mission goals. Finally, by collecting new, observable data, the sensors provide feedback to the reasoner about the effectiveness of its decisions and responses. If the new state, formed from the newly collected data, is better than the old state, the reasoner learns that the previous response was effective in moving the system closer towards achieving the cybersecurity and, ultimately, the mission goal. See figure 1.

Architecting a system to do all of the above is an enormous research challenge. We divide the work across several government laboratories, academic partners, and other government organizations inside and outside of NSA to create the foundational research, practical techniques, capabilities, and experiments to address the challenge. Our architecture must unify the results of these supporting research efforts into a single functional system to perform full-loop, autonomous reasoning and response. At the same time, this system must be initially designed with solid security principles, minimizing the chances that it can be exploited or subverted by an attacker, especially one employing adversarial machine learning (ML) methods.

Reinforcement learning: Overview

Reinforcement learning (RL) is one of the subfields of ML. It differs from other ML fields, such as supervised

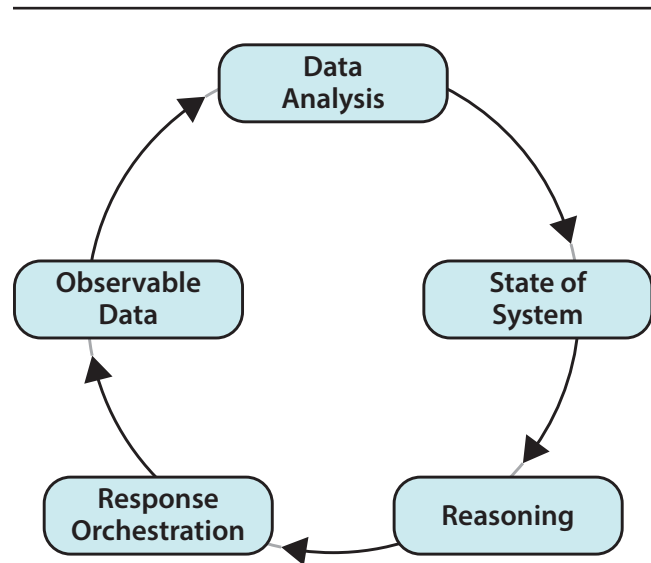


FIGURE 1. Monitoring feedback loop: Sensors collect observable data, data are analyzed to gain insight, system state is formed, the decision is made about optimal response to mitigate attacks, and a response is implemented. Sensors collect more information and next state is formed, providing feedback to reasoning process about impact on mission, service, or goal.

and unsupervised learning, because it involves learning how to map situations to actions in order to maximize a numerical reward signal [2]. RL is founded on the idea that we learn by interacting with our environment, which is a foundational idea underlying nearly all theories of learning and intelligence [2]. Thus, an RL learner, or agent, discovers on its own the optimal actions to achieve its goal based on immediate reward signals from the environment. In fact, actions taken in one state of the system may affect not only immediate rewards, but also all subsequent rewards. Thus, trial-and-error action search and delayed reward are two of the most important distinguishing features of RL [2]. Another interesting feature is the trade-off between exploration and exploitation [2]. An RL agent must exploit situations it has already learned from experience that produce rewards, but also explore new situations for possibly better rewards (see figure 2).

Beyond the agent and environment, an RL system features four main elements: policy, reward function, value function, and optionally, a model of the environment. A policy is a mapping from observed states of the environment to actions to take when in those states [2]. A reward function defines the goal of the RL problem. It maps each state, or state-action pair,

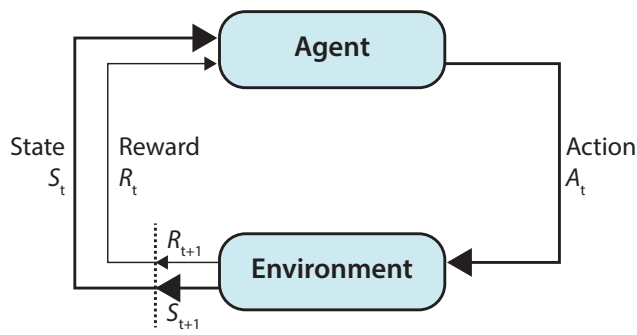


FIGURE 2. Given a state, the agent chooses an action that transitions the system to a new state and generates a reward from the environment. By accumulating these rewards over time, an agent learns the best actions for a given state or situation [2].

to a single number (i.e., a reward) indicating the immediate, intrinsic quality of that state [2]. By contrast, the value function specifies the long-run goodness of a given state. The value of a state is the cumulative amount of reward an agent can expect to receive over the future, starting from that state [2]. Finally, a model basically predicts the next state and reward from the given state and action. While in some dynamic, complex environments, such as computer networks, a sufficient model may not exist, RL can still be effective in solving problems in those “model-free” situations.

Reinforcement learning for autonomous cyber defense: Experiment

Our goal is to train a single RL agent to defend a sample enterprise network of 18 nodes (see figure 3) from being compromised by a cyber adversary. Each node has the same level of importance to a given mission and the same types of vulnerabilities. Achieving our goal in this simple scenario will provide proof-of-concept that RL can be used to develop an autonomous cyber-defense system. Recent advances in ML and RL have not focused on reasoning and response for autonomous cyber defense. Most current advances aim to improve detection of cyberattacks instead of reasoning about the best response to a cyberattack.

Thus, we must transform the autonomous network defense scenario into a RL problem. We make the standard assumption that the RL agent learns using a Markov decision process (MDP), which models the

random transitions between pairs of states. In an MDP, M is formally characterized as a 5-tuple, so $M = (S, A, R, P, \gamma)$, where [3]:

1. S is a set of states, with individual states denoted with lowercase s ;
2. A is a set of actions, with individual actions denoted with lowercase a ;
3. R is a reward function, written as $R(s, a, s')$ to indicate the reward gained from being in state s , taking action a , and moving directly to state s' ;
4. P is a transition probability function, written as $P(s'|s; a)$ to represent the probability of being in state s' after starting in state s and taking action a ;
5. γ is the discount factor, which represents how much future rewards are valued, γ has a value in the interval $[0,1]$. If γ is close to 1, then future rewards are highly valued. If γ is close to zero, then future rewards are not valued at all.

The MDP formulation allows us to find a high-quality policy function $\pi: S \rightarrow A$. A policy π is a function that, for each state, identifies a corresponding action to perform. In short, a policy, π , is a probability distribution function that maps states to actions. The optimal policy π^* is one that, if followed, maximizes the expected sum of discounted rewards [3]. To reduce the initial scenario complexity, we assume all nodes are equal with the same level of importance to the mission.

The most common class of RL algorithms is based on Q-Learning. These algorithms utilize state-action values denoted as $Q(s, a)$, which represent the

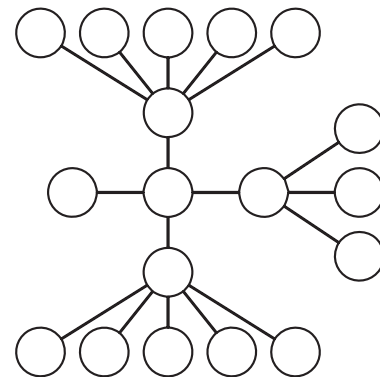


FIGURE 3. Enterprise network configuration with 18 nodes [4].

expected sum of discounted rewards when the agent starts at state s , takes action a , and from thereafter, chooses actions optimally. Once $Q(s, a)$ is known, the optimal action for each state is the action that maximizes $Q(s, a)$. By choosing the optimal action at each state to reach a given goal, the RL agent forms an optimal policy π^* .

The RL agent must learn the maximum $Q(s, a)$, or $Q^*(s, a)$, values. In standard Q-Learning, this is accomplished by first initializing all of the $Q(s, a)$ values at random. Then, the agent explores the state space, according to some policy. The most common of these is a *greedy-epsilon policy*, where epsilon, ϵ , represents some small probability that the agent will choose a random action at a given state, instead of the best, or greedy, action. This provides some guarantee that the agent, during its training phase, explores a large enough number of states such that its learned policy works sufficiently well during the testing phase on new states. This learning generalization, or transfer learning, problem is much studied in all fields of ML. So, at each time step the agent will take a random action with probability ϵ , or pick the current best action for the state with probability $1 - \epsilon$. The “current best action” is only “best” with respect to the current $Q(s, a)$ values; later iterations of the algorithm can cause $Q(s, a)$ value changes that would alter the best action when the same state s is encountered in the future [4].

Also, at each time step, the agent will receive a sample “state, action, reward, next state”-tuple, namely (s, a, r, s') from its environment. These samples are used to update the $Q(s, a)$ values using a moving average update [4]. Eventually, the updated $Q(s, a)$ values will converge to the optimal $Q^*(s, a)$ values, under some theoretical assumptions [3].

For our experiment, we assume that a given node can take on the following assignments or conditions [4]:

- ▶ **Assignment 0.** This represents being safe and non-isolated. The RL agent receives positive rewards for this node type.
- ▶ **Assignment 1.** This represents being compromised by an attacker and nonisolated. Since the node is nonisolated, the attacker could use it to spread to adjacent nodes.
- ▶ **Assignment 2.** This represents being compromised and isolated. The node does not represent a further threat to adjacent nodes.

Thus, our state of the network, and thus MDP, consists of the current node assignment of all 18 nodes simultaneously. Thus, the size of our state space is 3^n where $n = 18$. This implies that our state space is extremely large, with $3^{18} = 387,420,489$ possible states, even for a (relatively) small enterprise network size. In such large state space cases, the state-action value function, $Q(s, a)$ will be approximated, using a linear combination of a set of network features, instead of directly computed to enhance the speed of the Q-Learning algorithm. We used features such as the number of 0, 1, or 2 nodes, longest path of 0, 1, or 2 nodes, and maximum degree of any 0 node in our example [4]. Choosing the most important features for a state-action value function, or so-called feature extraction, is another challenging problem in all fields of ML. These features can be computed based on the sensor data collected and analyzed in the feedback loop described earlier.

We trained our RL agent using the following three actions:

1. **DO NOTHING:** At any given time step, the agent can choose to do nothing. Although there is no immediate cost to the agent for this action, the action can become costly if the adversary has compromised a node and spreads to neighboring nodes before the agent takes action to fix a compromised node.
2. **ISOLATE:** The agent disconnects a compromised node from the network. This action incurs an immediate cost, and is more expensive than PATCH. It also costs the agent another time step to reconnect the node.
3. **PATCH:** The agent fixes a compromised node, whether or not it is isolated. If the node was isolated, then this action returns the node to safe status. If not, the compromise has some chance of spreading before the agent completes the fix. This action is less expensive than ISOLATE, but could ultimately be more costly depending on amount of attacker spreading.

Ideally, our agent should use RL to learn the cost-benefit trade-off between choosing the ISOLATE and PATCH actions. Instead of using a predefined set of static decision rules to determine the correct action, the agent learns the best actions by trial-and-error based on the rewards received from the environment. In our example, the reward is the difference between

the number of uncompromised nodes and the cost of the current action. In other words, $R(s, a, s') = N(s) - C(s)$ where $N(s)$ is the number of uncompromised nodes in the current state s and $C(s)$ is the cost of the action performed in state s . We used the numerical values of 10, 5, and 0 for the cost of the isolate, patch, and do nothing actions, respectively [4].

The state transitions in our MDP are stochastic, because there is no certainty about which node the attacker will compromise next. Thus, when the agent chooses an action, the next state is not known with certainty. Our MDP is effectively modeling a two-player game. At each turn, the adversary has two options to influence the network state:

1. **Spreading:** For each unsafe (i.e., compromised) and non-isolated node, the attacker has some probability p of also compromising an adjacent node that is safe and independent of other nodes. For instance, suppose $p = 0.25$ and the network contains only three connected nodes, where only one is unsafe. Then the attacker has a 0.252 chance of compromising the other two nodes, a 0.375 chance of compromising exactly one of the adjacent nodes, and a 0.752 chance that none of the other nodes are compromised [4].
2. **Random Intrusion:** For each node, the attacker has some probability of randomly compromising

it, independent of the other nodes. For example, if the probability is 0.1 and the network state has three nodes that are all safe, then there is a 0.13 chance that the attacker manages to get in all the nodes at once. This action is applied during each attacker's turn after it has attempted to spread as described previously [4].

These two actions come with different probabilities. In general, we set the probability of the intrusion spreading to a node adjacent to a compromised one to be much higher than the probability of a cyberattack, i.e., intrusion, compromising a random node. With the network of 18 nodes, a good starting value is 0.15 for the intrusion "spreading" probability, and 0.01 for the probability of random compromise. These probabilities will have to be modified appropriately as the number of network states changes, since we assume that the RL agent can only act on one node per turn. If the spreading probability is too high, the RL agent effectively cannot protect the network even if it performed the optimal actions [4].

Reinforcement learning for cyber defense: Initial results

To evaluate our agent's performance for a single episode of 100 turns, we compute the number of safe nodes (out of a total of 18) after each turn, and then

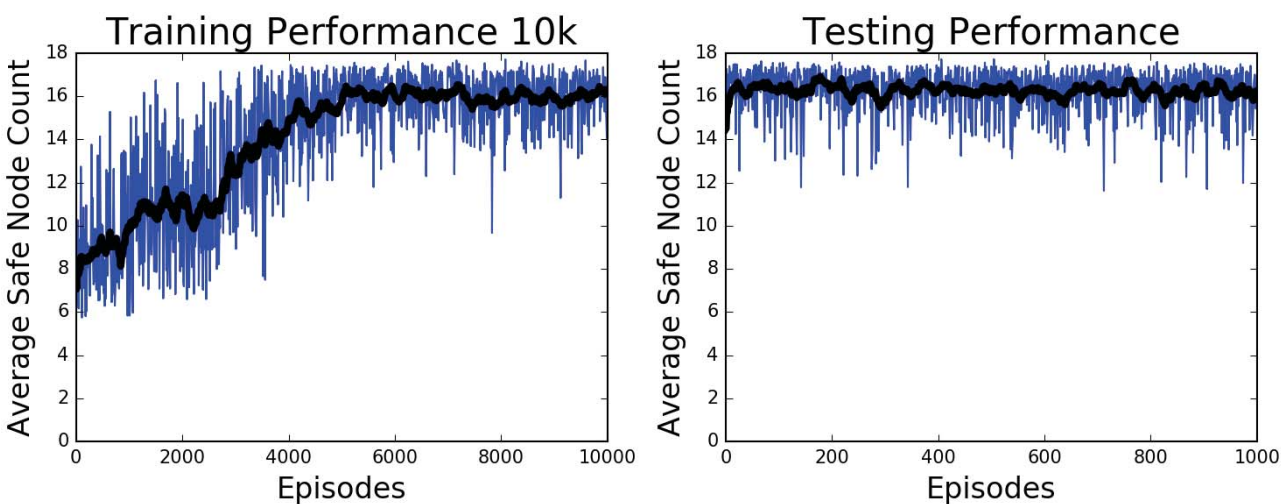


FIGURE 4. The performance of an RL agent defending the network autonomously, measured in terms of the average number of safe nodes out of 18. The left plot shows training with the annealed greedy-epsilon policy; the right plot shows testing with fixed $\epsilon = 0.05$. We use 10k and 1k training and testing episodes, respectively. The black curve is a smoothed version of the blue curve [4].

average those numbers across all turns. The average number of safe nodes is a performance metric that accounts for the entire episode of 100 turns and should smooth out unlucky turns when the attacker’s “random intrusion” compromises many more nodes than expected. In the RL literature, these episodes are also known as “rollouts” or “trials” [2]. See figure 4.

To evaluate the RL agent’s overall performance, we compute the whole sequence of average number of safe nodes for each test set episode. Note that because of the attacker’s intrusion ability, the odds of an agent scoring a perfect 18 for any one episode are exceedingly remote. For instance, based on our chosen probability of compromise and probability of spreading values, the attacker has basically a zero probability, i.e., $((0.98^{18})^{100} \sim 0$, of never compromising any node during every turn of one episode. Therefore, an agent has basically a zero probability of keeping all 18 nodes safe during every turn [4].

For our testing the performance of our RL agent, we used a standard value of $\epsilon = 0.05$. However, other choices of ϵ may be better for our RL agent, and we would also like to find a suitable range of ϵ values that reaches consistent testing performance. Figure 5 shows our results, starting from the top left and

moving clockwise, for ϵ in the set $\{0.0; 0.05; 0.1; 0.2; 0.4; 0.6\}$ [4]. In figure 5, we recomputed the testing performance results for the same $\epsilon = 0.05$ value used in figure 4 as a sanity check. The $\epsilon = 0.0$ case means we never randomly choose actions. In other words, we only choose the best actions for the given state. We see that the three smaller ϵ values result in similar performance. The quality of RL agent noticeably decreases with $\epsilon = 0.2$, and worsens, as expected, as ϵ increases. The worst performance is obtained when ϵ reaches 1, representing the agent always choosing a random action for a given state. Figure 5 displays the exact performance values that are best for our example and indicates that $\epsilon = 0.05$ is reasonable for our greedy-epsilon policy [4].

We now attempt to achieve the best test performance possible. Using the information learned from previous sections, we maintain the greedy-epsilon policies for training and testing and do not use weight normalization. We also run Q-Learning for 200,000 trials, a 20-fold increase over our previous number of trials.

Our RL agent results are shown in figure 6 in terms of performance. The performance plots show that the agent has successfully learned how to control the

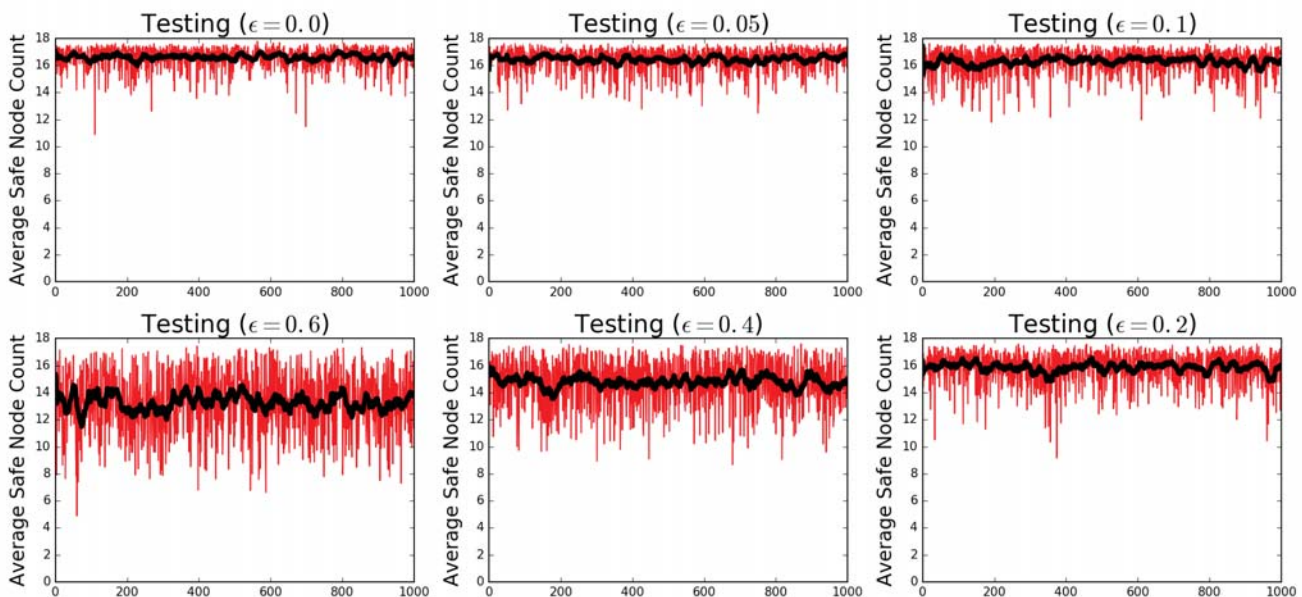


FIGURE 5. Testing performance plots for various values of ϵ . All six subplots were generated using performance results computed across 1,000 total episodes [4].

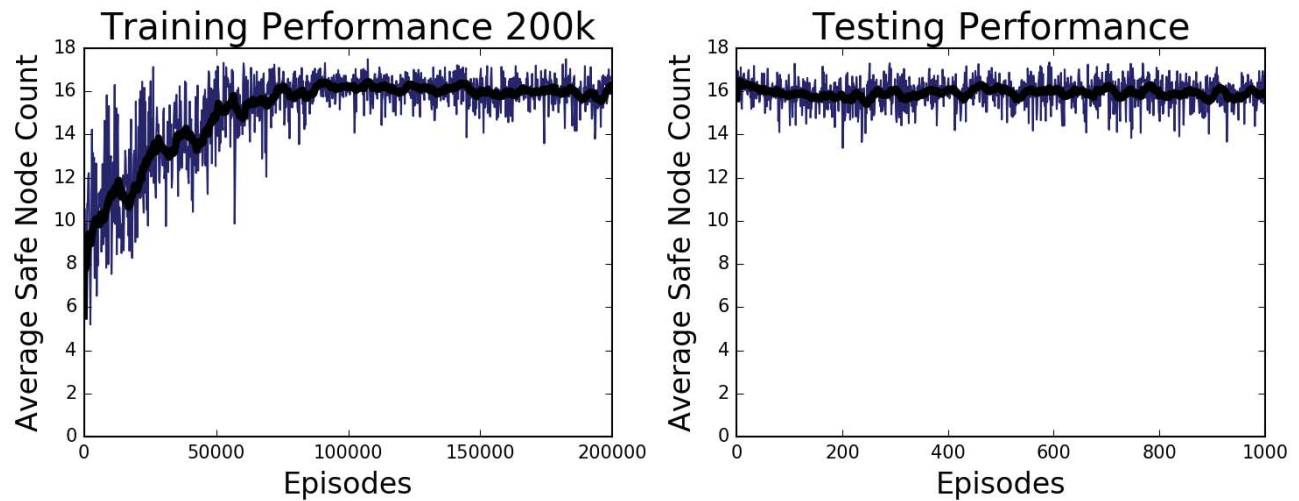



FIGURE 6. The performance of the RL agent measured in terms of the number of safe nodes out of 18. The performance is similar to that presented in figure 4, except with more (i.e., 200k) training episodes [4].

network to a large degree. The average value over all 200,000 trials is 15.919, with a standard deviation of 0.650. Despite all the training trials, however, the basic experiment with 10,000 training trials actually performed better with a mean score of 16.265 (though with a higher standard deviation). The best performing version of the RL agent uses $\epsilon = 0.0$ (average value of 16.600), though we caution that this is likely because of the limited actions in this scenario. In a more complicated domain, such as Atari 2600 games, playing random actions in practice might be necessary.

Conclusions

We demonstrated a nontrivial computer network defense scenario and trained an RL agent that successfully protects the network from a randomized adversary. Moreover, our agent has to reason when selecting actions, because they are designed to have competing trade-offs between safety and cost. We presented an 18 node scenario, using a plain graph where all nodes had equal value. As evidenced by the results over many testing episodes, the agent consistently keeps a significant number of nodes safe. However, the primary weakness of this scenario is that it assumes very simplistic models for the attacker and defender. The most obvious direction for future work is therefore to design more realistic cyber attacker-defender models.



Moreover, we showed that RL can be used to develop an autonomous cyber defense system that improves the resilience of an enterprise network to cyberattacks. As noted, more research is required to show that RL can be effective in larger, more complicated enterprise network configurations, where nodes vary in type and potential vulnerabilities, and in environments with sophisticated attacker behavior. As an initial research study, though, the results are promising. 

References

- [1] Bodeau D, Graubart R, Heinbockel W, Laderman E. “Cyber resiliency engineering aid—The updated cyber resiliency engineering framework and guidance on applying cyber resiliency techniques.” (Technical report MTR140499R1). May 2015. The MITRE Corporation.
- [2] Barto AG, Sutton RS. *Reinforcement Learning: An Introduction*. Cambridge (MA): MIT Press; 1998.
- [3] Beaudoin L, Japkowicz N, Matwin S. “Autonomic computer network defence using risk state and reinforcement learning.” *Cryptology and Information Security Series*, 2009.
- [4] Seita D. “DanielCanProtectIt: An AI agent that learns how to defend a computer network system using reinforcement learning.” (Technical Report). August 2016. NSA Research Directorate.

Rethinking neuromorphic computation: Why a new paradigm is needed

by Mark R. McLean and Christopher D. Krieger



More than 80 years ago, Alan Turing developed a computational paradigm that enabled machines to perform operations humans are not adept at—fast, precise math. Today, highly optimized central processing units (CPUs) and graphics processing units (GPUs) can do trillions of precise mathematical operations per second. Now researchers are trying to use these processors in a different way—to do tasks that humans *are* good at. This field of research is called machine learning (ML) or artificial intelligence (AI), and their associated algorithms are providing these new capabilities. This article describes our search to identify extremely efficient hardware architectures that execute ML algorithms, and how it ended up leading our research in an unanticipated direction.

[Photo credit: monsitj/iStock/Thinkstock]

ML applications deviate from the traditional computer programming methods in that they learn from data. They learn so well, in fact, that they provide better solutions for some tasks than our smartest programmers. Face recognition, speech recognition, and even autonomous driving are just a few applications for which ML algorithms surpass human ability to directly program a better solution. As a result of the remarkable capabilities ML provides, billions of dollars are being spent by industry to develop new or modify current processors to provide more efficient computation of ML algorithms. These ML-focused computational processors are more commonly referred to as *neuromorphic processors* (NMPs). Realizing the capability NMPs could provide to NSA, the Laboratory for Physical Sciences (LPS) launched the Neuromorphic Computation Research Program (NCRP).

Evaluating neuromorphic processors

The primary goal for the NCRP is to explore novel NMP architectures that could provide revolutionary computational efficiency, which we define as being greater than two orders of magnitude improvement in classifications/watt. In pursuit of this goal, we have explored a large number of the current and proposed NMPs from nearly all the major processor vendors. These NMP designs use similar complimentary metal-oxide semiconductor (CMOS) process technologies, so they rely on architectural design changes to provide a competitive advantage.

One of the NMP architectures we evaluated was the Google tensor processing unit (TPU) [1]. The TPU is an application-specific hardware design that focuses on providing efficient inference for Google's ML services. Recently, NVIDIA compared the TPU and the NVIDIA P40 GPU to better evaluate the performance of both designs. Directly comparing these NMPs is difficult because the two products were designed for different purposes; the P40 was designed as a more general-purpose computational system than the TPU. Considering this, the inference comparison shows the TPU to be only 6.3 times more power efficient than the P40. While this efficiency gain is very respectable within our highly optimized computation domain, it is not close to our desired revolutionary efficiency gains.

Moreover, it demonstrates that even with an entirely new application-specific design, the TPU could

not surpass a general-purpose GPU efficiency by a single order of magnitude (see figure 1).

We also evaluated the IBM TrueNorth processor, which was developed under the Defense Advanced Research Projects Agency's Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) program. This processor is unique because it simulates spiking neurons and is supposed to be very power efficient. In our evaluation, we compared a low-power GPU, the NVIDIA Jetson TX2, to TrueNorth and ran the same benchmarks as the IBM TrueNorth paper [2]. When all the power required to make a deployable system (e.g., random-access memory, complex programmable logic device) was taken into account, the TrueNorth processor was no better and sometimes even less efficient than the inexpensive NVIDIA Jetson TX2.

We concluded that spiking networks, using current algorithms, were not likely to provide revolutionary efficiency. Additionally, we evaluated a number of future architectural designs from leading computer processor companies, and we expect them to provide steady but incremental improvements in efficiency. The evaluation of this broad range of NMP designs indicates that evolutionary architectural changes will not provide a disruptive efficiency benefit.

Evaluating memristors

As a result, we decided to explore nontraditional computing approaches. We spent three years conducting in-depth evaluations involving a radically different way to perform multiply and accumulate operations (MACCs) [3]. Since the MACC operation is a fundamental computation for a number of neural networks, the expectation was that an efficiently computed MACC operation might provide our desired improvement. The research focused on the use of memristors connected in a crossbar, which is the focus of many research groups [4]. Memristors were theorized by Leon Chua in 1971 [5] and arguably discovered by Hewlett Packard's Stan Williams in 2009 [6]. Memristors are a two-terminal device that if given enough energy can increase or decrease their resistance; while at lower energies, they can be read without resistance change.

Our interest in memristors was fueled by wanting to determine if they could replace the synaptic weights of a neural network. When applying low voltages,

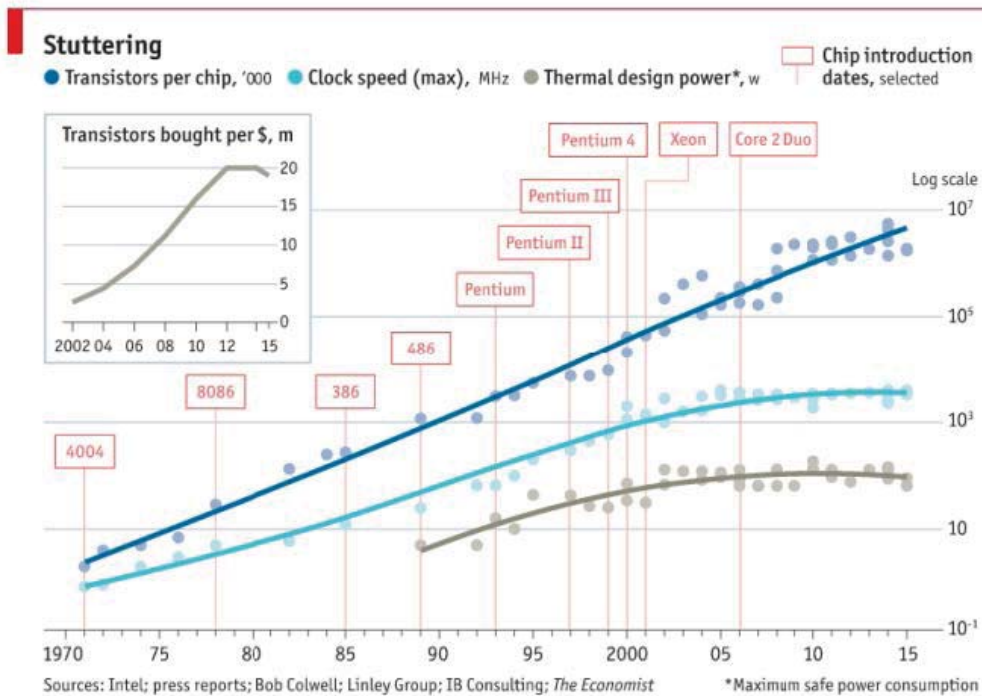


FIGURE 1. Hyper-optimized processor space is constraining computational advancements. [Figure credit: Oliver Mitchell, Robot Rabbi blog, <http://robotrabbi.com>]

memristors act like fixed resistors, and by leveraging Ohm's law, a multiply operation can be performed. The accumulate operation would be performed by having all the memristors connect to a single metal line in a wired "OR" configuration. Combining these devices into high-density memristive crossbars would enable billions of MACC operations to be computed in parallel.

In the summer of 2016, our team conducted an analysis to test the benefits of memristor-based crossbar computation. We created a simulation program with integrated circuit emphasis (SPICE)-level netlist of a two-layer neural network that evaluated an MNIST-trained model; MNIST (Modified National Institute of Standards and Technology) is a simple and standard ML data set that has images of handwritten digits from 0 to 9 represented in a 32 by 32 array. For baseline comparison, we also designed a SPICE netlist for a CMOS adder-based neural network. Both netlists provided inference capability and used the same weights from offline training. After completing multiple SPICE simulations of both architectures, we found that using memristors for computation would

provide approximately a four times improvement in compute efficiency (i.e., classifications/watt); unfortunately, this was nowhere near our goal. Furthermore, the four times improvement was derived from the computational efficiency in isolation, but a computational system must also expend energy on input-output communication (I/O).

To compare the efficiency benefit that memristors provide at a system level, we calculated the energy required to transfer information to the memristive crossbars. For our computation, we used one of the most efficient memories available (i.e., Micron's Hybrid Memory Cube), which consumes only 10 picojoules per bit. These calculations revealed that the system-level efficiency provided by memristors would result in only a 1% power savings. Evaluating the system-level efficiency highlighted the dependency of I/O and also emphasized that it is the largest energy consumer. Putting this into perspective, in a presentation at the 2011 International Conference on Parallel Architectures and Compilation Techniques, Intel Fellow Shekhar Borkar showed that for the LINPACK benchmarks, processor I/O consumes two

thirds of the system power, and compute consumed only one third [4]. The implication is that—even if computation energy could be reduced to nothing—the resulting system-level efficiency would be improved by only 33%. This revealed a hard constraint: To make a revolutionary impact on power efficiency, *both* I/O and compute power need to be orders of magnitude more efficient.

We decided to step back and look at the facts. We had explored the broad domain of current and future NMP designs, and we did not find any that had the potential to meet our efficiency goals. Furthermore, current computation advances are also being constrained by a number of physics limitations, just one example is scaling current CMOS technologies. Transistors will likely not scale down much further, but even if they could, thermal density will limit the number that can be used. We also explored more novel computation using memristors and found they provide no substantial advantage. Additionally, we realized we couldn't just focus our search on efficient computation; our solution had to also include efficient I/O.

While we were discouraged by our inability to find a solution, our evaluations provided further insight into defining what a solution would require. We are fairly confident that a solution can exist. Why? Because the human brain provides a tangible realization of our goal. While examining the constraints that the solution would require major efficiency advancements to both compute and I/O, we wondered if ML tasks could be accomplished using multiple orders of magnitude less compute and I/O operations. This would enable us to meet our efficiency goals. But was there an underlying principle that could provide this reduction?

Inspired by the brain: Computing on concepts

Our search for a solution led us to look at the one of the most useful ML algorithms, called a convolutional neural network (CNN). CNNs were developed by Yann LeCun [7], and variations of these networks have held the highest marks for accuracy on a number of

standard ML data sets. An aspect of CNNs of particular interest is that they employ a deep hierarchy of computational layers. To understand the functionality hierarchy provides, we leveraged research from Jason Yosinski at Cornell University [8]. Yosinski and his team developed a deep visualization toolkit that enables visualization of what the different layers in the hierarchy learn. There is a short video of this on YouTube¹ showing that as information propagates through the hierarchy, each consecutively higher layer is learning a more abstract representation of the previous layer. For example, at the lower layers, the CNN is learning edges, the next layers are learning shapes and parts of objects, and by the fifth layer, individual neurons are representing abstract concepts like faces, dogs, and text.

If computations could be performed on these concepts rather than all the bits of information that they represent, it would significantly reduce both compute and I/O. Our preliminary investigations indicated that *computing on concepts* could provide a mechanism to significantly increase computational efficiency, but we realized that it could require the development of a new information processing paradigm that is inspired by the brain.

As a society, we have been trying to understand how the brain works for centuries, yet there is no unified theory of how the brain processes information. We have amassed so much data about the brain that it can be interpreted to support contradictory ideas, making knowledge extraction difficult. As a result, we are bounding our investigation of the brain to improve our chances of success. Our focus is on understanding the high-level information processing paradigm used by the brain. While this is certainly a lofty goal, it does attempt to remove the biological implementation details. We are emphasizing the functionality—how the brain transforms information—and attempting to leave behind the complexity of how it does it. While it is impossible to completely overlook implementation, staying as removed from it as possible will improve our ability to piece together this paradigm. This approach has enabled us to extract a number of principles that we feel are essential elements of the brain's information processing paradigm.

1. See <https://www.youtube.com/watch?v=AgkflQ4IGaM>, or just search for deep visualization toolkit.

Brain evolution

Before discussing these principles, it is important to discuss the mechanisms that guided the development of our brain. Let's start by describing some of the mechanisms and constraints that led to the existence of our current human brain. The brain as an information processing system has taken millions of years to evolve. Evolution progresses from the success of its previous generation; it can't scrap a design and start from scratch like Google with their TPU. Evolution's main optimization constraint is survival of the organism's genetic code. Survival is the ability of the organism to get the required energy to keep its life processes functioning and enable procreation. This puts considerable constraints on what the organism expends energy on.

Evolution, moreover, is not directed and is kept in check by our physical environment. For example, dinosaurs emerged by evolution placing a priority on size and strength and not on intellect. When the Earth's environment changed rapidly, the dinosaurs didn't have enough intelligence to adapt to the situation, and a large percentage of them died. This exemplifies the interaction between evolution and the environment; the environment can scrap bad designs. When this rapid environmental change occurred, organisms that could adapt survived and procreated, and this is how evolution and the environment started to favor organisms with more intelligence.

However, evolution is constantly balancing the required intelligence of an organism against the energy it consumes. Additionally, there is no evolutionary motivation to improve intellect beyond the organism's survival. It is the competition between other animals that continued to push intellectual improvement. Even with this competition, evolution is slow and incremental; reusing features that work and constantly balancing complexity with the energy it requires. This is the optimization landscape that our brain evolved under, and it provides an essential foundation for unraveling its information processing paradigm.

Brain information processing paradigm

Our goal is to uncover the information processing paradigm implemented by the brain, and as such,



FIGURE 2. Silhouettes provide critical, spatially correlated information enabling object recognition regardless of scale. [Image credit: Vecteezy.com]

we have to define the information it processes. Our brain did not evolve to process numbers, so what does it process? To answer this we need to look at our environment. Through evolution, organisms had to survive in our world. Our world resides in three-dimensional space and time, and our brains evolved to efficiently process information and survive within this space. A few of the main brain functions that developed were object recognition and prediction. Objects need to be recognized at different distances and in different lighting conditions.

In our world, all objects are three dimensional and contiguous, and the largest feature of an object is its outline or silhouette. To illustrate how much information these structures contain, see how easy it is for us to recognize objects only by their silhouette (see figure 2). There is no real color information; there is just a two-dimensional representation that contains relative structural information. The silhouette is an example of spatially correlated change (i.e., edges), which is a type of information our brain processes.

Support for this idea comes from the functionality of our retina. At the input to our visual cortex, the retina removes redundant information and extracts the spatial structure of the object; a simplified functional description is that it is an edge detector. From the very start of information processing, edge detection greatly reduces the amount of information our

brain must compute and communicate. The retina contains approximately 100 million rods and cones, while the optic nerve has only about one million neurons [9]. The retina reduces the amount of information reaching our brain by two orders of magnitude. This has profound implications on how reducible natural signals are and provides valuable insight into what information is important. Vision is just one of our senses that provide information to the brain, and creating relationships between different senses may be accomplished using temporally correlated events. If you see a change and you hear a noise at the same time, you tend to link the events together. Spatially and temporally correlated events are the structured information our brains process.

The requirement to process simultaneous spatial and temporal events provides strong support that our brain uses spatial computation and is event driven. Moreover, the retina passes edges, and these edges are spatially correlated change—this implies the brain computes on changes. Spatial computation, event-driven processing that only computes on change creates a completely adaptive power-saving mechanism. Energy is consumed only when there is information to be processed and then only the locally affected neurons use energy to process it. That being said, if the brain is an event-driven architecture, then it also requires a method of prioritizing the processing of these events.

The priority for events should be directly related to the information the event contains; this leads us to believe that new information and rapid spatial change are events that contain high-priority information. It also follows that these events would have higher spiking frequencies and consume more energy. These higher firing rates could also be used to direct our attention. The brain is always trying to minimize energy consumption and uses our attention as a tool to achieve this. Processing the high-frequency events first minimizes energy and prioritizes the information we process. A tangible example of this principle is how a magician commands your attention by making large gestures with one hand while he makes the subtle switch with his other hand. The large gestures evoke high spatial change on the retina and cause more neurons to fire; thus, the large gestures draw your brain's attention in an attempt to minimize its energy consumption.

We previously mentioned that computing on concept is a foundational principle of the information processing paradigm of the brain. We have seen from the deep visualization toolbox that CNNs create abstract concepts as information propagates up the hierarchy. The idea of using a hierarchy to process information is very different compared to our current computational paradigm. There is considerable research supporting the idea that the brain processes information in a hierarchy [10, 11]. However, unlike the CNN structure, we believe each layer in the hierarchy also connects to a central point. This point would be similar to the thalamus/hippocampus in the brain, and this additional connectivity is what enables the brain to easily use varying degrees of abstraction to accomplish tasks or recognize objects.

Prediction is another foundational principle of the information processing paradigm of the brain. Research on the inferior temporal cortex of monkeys shows how, after learning a sequence of patterns, the monkey's predictions minimize neural firing [12] (see figure 3).

Novel sensory information is passed up the hierarchy, and these features evoke a hierarchical prediction back down. As the prediction passes down the hierarchy, differences between the prediction and sensory input are calculated at each level. If these differences are small, then at some level of abstraction within the hierarchy, we “understand” what we are seeing. If we “understand” by choosing a good predictive model, then the firing rate of neurons—and hence the energy consumption of the brain—decreases. In addition, the difference between sensory input and prediction may play a crucial role in the learning algorithm. With a good prediction, there may not be enough available energy to learn, but with a bad prediction, there are still a number of neurons firing at a high frequency, and this additional energy may enable learning. Furthermore, this additional energy would then draw your attention to the prediction errors, which would be the highly discriminant information. An example of this is if a person has a unique feature on their face, we tend to fixate on it and learn to associate that feature to better identify the person.

We have described how both prediction and attention combine to minimize neural firing and thereby minimize energy consumption. In the visual cortex, there is a ventral (what) and a dorsal (where) pathway.

Information propagating through the “what” hierarchy is reduced into a concept and then associated in the “where” space. What we believe is happening, is that the brain is creating a personal reality. New objects are identified and added to this reality. As our attention moves over objects in our reality, predictions are propagated down the hierarchy and compared with our sensory information. Only the differences are communicated to the thalamus/hippocampus, and our personal reality gets updated at the level of abstraction that is needed to achieve the current task. The creation of a personal reality would provide the substrate to achieve our energy efficiency goals by utilizing the concepts we create with our hierarchy.

Conclusion

This article has summarized the NCRP’s search for highly efficient NMP architectures. From our broad evaluation of this computational landscape, we don’t believe it is possible for evolutionary design changes to meet our disruptive efficiency goals. Ultimately, the solution must significantly reduce both I/O and compute power, and computing on concepts may provide

that reduction. This hypothesis directed our research to the development of a different information processing paradigm that is guided by the way our brain processes information. As we attempt to develop this new paradigm, we have to be careful not to get stuck in biological implementation details and stay focused on how information is transformed as it propagates through the brain.

Initial investigations are leading us to explore the possibility that this paradigm creates a personal reality where higher-level computations occur; however, the exact nature of these computations is still being determined. We are just starting to develop this paradigm, and we hope within the next few years to develop an initial framework, have a more detailed analysis about potential energy savings, and define suitable applications. Due to the vast complexities of the human brain, this framework will only capture a small subset of human brain behavior. Motor control, emotions, motivation, and higher-level reasoning are just a few of the functionalities that we are not addressing.

A parting perspective is that the majority of NMP research uses a computational paradigm designed for

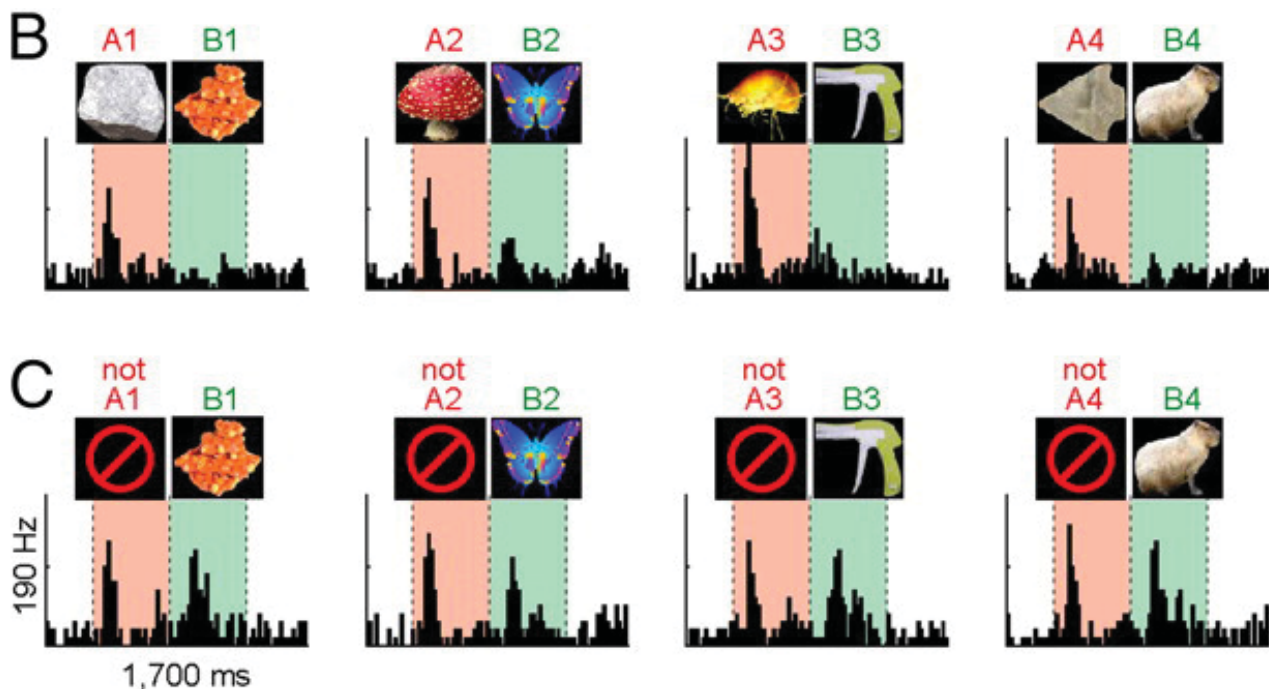



FIGURE 3. Row B shows neural firing reduction on trained sequences. Row C illustrates neural firing patterns on untrained object sequences. Figure from [12].

high-speed precise math—which humans are not good at—to *simulate* tasks humans excel at. Precise math can be used to accurately simulate physical systems, such as CPUs; however, this approach is very computationally inefficient compared to using the CPU itself. Without the guidance from a new computational paradigm, NMPs will essentially be relegated to being hardware-based simulators. We expect that our new paradigm will integrate the principles discussed in this article and likely require more we have yet to uncover. That being said, if we are successful, it is possible we will achieve more than just highly efficient computation. This paradigm could provide the blueprints for designing a system with the fundamental abilities to understand, seamlessly work with multiple levels of abstract concepts, and inherently extract knowledge from data. 

References

[1] Jouppi N, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A, et al. “Datacenter performance analysis of a tensor processing unit.” 2017. Cornell University Library, arXiv:1704.04760v1.

[2] Esser S, Merolla P, Aurthur J, Cassidy A, Appuswamy R, Andreopoulos A, Berg D, McKinstry J, Melano T, Barch D, et al. “Convolutional networks for fast, energy-efficient neuromorphic computing.” *Proceedings of the National Academy of Sciences of the United States of America*. 2016;113(41):11441–11446. Available at <https://doi.org/10.1073/pnas.1604850113>.

[3] Yokopcic C, Hasan R, Taha T, McLean M, Palmer D. “Efficacy of memristive crossbars for neuromorphic processors.” In: *International Joint Conference on Neural Networks*; 2014 Jul 6–11; Beijing, China. doi:10.1109/IJCNN.2014.6889807.

[4] Mountain D, McLean M, Krieger C. “Memristor crossbar tiles in a flexible general purpose neuromorphic processor.” In: *IEEE Journal on Emerging Technologies and Selected Topics in Circuit Systems*; 2017; PP(99):1–1. doi: 10.1109/JETCAS.2017.2767024.

[5] Chua L. “Memristor—The missing circuit element.” In: *IEEE Transactions on Circuit Theory*; 1971;18(5):507–519. doi: 10.1109/TCT.1971.1083337.

[6] Strukov D, Snider G, Stewart D, Williams S. “The missing memristor found.” *Nature*. 2008;453(1):80–82. doi:10.1038/nature06932.

[7] LeCun Y, Haffner L, Bottu L, Bengio Y. “Object recognition with gradient-based learning.” In: *Shape, Contour, and Grouping in Computer Vision*. Berlin, Heidelberg: Springer; 1999.p. 319–345. Available at: https://doi.org/10.1007/3-540-46805-6_19.

[8] Yosinski J, Clune J, Nguyen A, Fuchs T, Lipson H. “Understanding neural networks through deep visualization.” In: *International Conference on Machine Learning*; 2015 Jul 6–11; Lille, France.

[9] “Chapter 11, Vision: The eye.” In: Purves D, Augustine G, Fitzpatrick D, Katz LC, LaMantia AS, McNamara JO, Williams MS, editors. *Neuroscience 2nd edition*. Sunderland (MA): Sinauer Associates; 2001.

[10] Riesenhuber M, Poggio T. “Hierarchical models of object recognition in the cortex.” *Nature Neuroscience*. 1999;2(11):1019–1025.

[11] Lee TS, Mumford D. “Hierarchical bayesian inference in the visual cortex.” *Journal of the Optical Society of America*. 2003;20(7):1434–1448.

[12] Meyer T, Olsen C. “Statistical learning of visual transitions in monkey inferotemporal cortex.” *Proceedings of the National Academy of Sciences of the United States of America*. 2011;108(48):19401–19406.

[13] Borkar S. “The exascale challenge.” In: *20th International Conference on Parallel Architectures and Compilation Techniques*; 2011 Oct 10–14; Galveston Island, TX.

[14] Huth A, Nishimoto S, Vu A, Gallant J. “A continuous semantic space describes the representation of thousands of object and action categories across the human brain.” *Neuron*. 2012;76(6):1210–1224. doi: 10.1016/j.neuron.2012.10.014.

[15] Meyer T, Olsen C. “Statistical learning of visual transitions in the monkey inferotemporal cortex.” In: *Proceedings of the National Academy of Sciences of the United States of America*. 2011;108(48):19401–19406. Available at: <https://doi.org/10.1073/pnas.1112895108>.



HOW

deep learning

CHANGED COMPUTER VISION

by Bridget Kennedy and Brad Skaggs

If you are aware of the term *deep learning*, you are likely also aware of where deep learning has been most disruptive—that is, the field of computer vision (CV)—the study of teaching a computer to identify the content of an image or video. Problems in this area range from facial recognition to object recognition, scene understanding, geolocation of an image given only the objects in the image, and three-dimensional reconstruction from two-dimensional images. The majority of CV research is focused on imbuing a machine with a sufficient level of recognition so as to automatically sort large quantities of images and videos by their visual content.

Deep learning has been transformative in CV for a number of reasons, and the speed with which CV went from what one might perceive as primitive analytics to highly sophisticated capabilities as a result of leveraging deep learning offers lessons for other domains seeking to produce game-changing analytics in new areas.

Before deep learning

CV is a subfield of computer science. It has a past that predates the current excitement surrounding the apps that allow you to turn a picture of yourself into a Picasso drawing [1] and tools that allow you to search and sort your pictures by visual concepts such as cats, children, parties, and so on [2, 3, 4, 5].

What did people used to do before deep convolutional neural networks disrupted the field? Understanding this process may help to understand the shift in design that was allowed by the use of deep nets.

The standard procedure for analyzing an image or the pixels within a video frame starts by running an algorithm that looks for “interesting” local regions within an image (i.e., local pieces that stand out from the rest—corners, edges, bright or dark regions with strong visual gradients). After interesting points are located, one describes these local regions using feature vectors. Some common image feature vectors, such as SIFT, SURF, BRIEF, ORB [6, 7, 8, 9], and so on, are handmade and were painstakingly developed by experts in the field. In designing these feature vectors, researchers performed years of research to ensure that their features were robust and unaffected by changes in lighting, scale, and rotation.

After descriptions of the local areas in an image are made, a process of dimension reduction, pooling, clustering, and/or classification of these regions are performed, allowing one to describe an image both locally and generally. The general classifications are the most important for semantic labeling tasks, such as teaching a computer to recognize that an object is a cat. These methods, and the overall flow from pixels to classes, were optimized and improved, and the field was incrementally moved forward one PhD dissertation at a time.

The goal of all this work was often the same: How do we teach computers to see as a human being sees? How do we teach it to go from pixels to cats? The process of going from low-level features to semantic or high-level concepts was still being thought out by researchers working with features such as SURF when deep convolutional neural networks became ubiquitous.

What does *deep* mean?

The power of deep learning for CV comes from the ability of a deep network to learn features in a layered fashion, ranging from edges and gradients to semantic concepts like ears, faces, writing, and so on. No longer would one need to spend years coming up with the best way to detect and describe local regions of interest or to determine how best to glue these low-level ideas together to approximate semantic ideas. Instead, one could *merely* push millions of images through a deep network structure, and learn the optimal way to separate these images into classes using a target classifier and fast learning algorithms. The deep network would learn how to transition from the low-level to the semantic. This is really critical. Deep learning eliminated the need to hand design feature vectors at the same time it learned how to build semantic feature vectors, thus changing the way the community works. The mantra became: Learn features directly from your data—never design a feature vector by hand.

One of the most interesting pieces of a deep convolutional neural network trained on CV tasks—such as those trained on the large ImageNet [10] corpus with 1,000 classes and over a million images—is that the semantic feature vector encodings of images can be recycled. That is, even if a particular category wasn't present in the original training categories, there are enough high-level (semantic) categories pulled out or learned in the process of building the deep networks for ImageNet classification, that these feature vectors can be quickly transferred to new tasks. For example, one of the basic examples when learning how to use TensorFlow [11] is a *transfer learning* task, where one uses the deep features learned on ImageNet to quickly learn a classifier on a small data set. The typical first transfer learning task is one to classify flower types by training on the Oxford Flowers data set [12], but an extremely effective bird classifier is also easily obtained using Caltech Birds (CUB-200) [13] and can be learned in minutes.

There are many arguments about what makes something deep learning. Is it the size of the network? If so, how deep is deep? ResNet-101 [5], where 101 is the number of layers, was the standard net for about a year—the life span of a reigning champ deep net. A

better rule of thumb should be how well do features learned in the training process transfer to different but related tasks—like the easy transfer of the deep Inception features trained on ImageNet to a general flower classifier. Did your network learn shallow or deep concepts? Low-level or semantic concepts?

Applying deep learning in other fields: Three lessons from CV

While some of the dramatic improvements that deep-learning CV applications have seen might be a result of the unique nature of CV problems, there are certainly lessons to be gleaned for achieving success in other fields beyond vision.

Adopt standard, sharable data sets

The CV community has a de facto ladder of standard data sets of increasing complexity. The most frequently used data set for basic testing of a model architecture for image classification is the MNIST (Modified National Institute of Standards and Technology) database of handwritten digits [14]. An algorithm successful on MNIST might next be validated on the CIFAR-10 or CIFAR-100 (Canadian Institute for Advanced Research) [15] data sets with 10 or 100 object categories more complex than digits. The final rung of the ladder is testing on ImageNet, which has roughly 14 million images (one million with bounding boxes) in 1,000 categories [10].

For an approach claiming state-of-the-art in image classification, reporting results on at least the upper rungs of the MNIST/CIFAR/ImageNet ladder is necessary for publishable work in image classification. In fact, even nonclassification tasks often leverage data derived from these standard data sets.

For a field to adopt common evaluation data sets, practitioners must address the legal complexities of licensing large, diverse data sets derived from many sources. ImageNet handles this problem by providing URLs pointing to known locations of the images; it is up to each research team to download them for themselves, or else download them from the ImageNet curators after agreeing to several restrictions on use. As an example from the text processing community,

a purpose-built question/answer data set was built by crowdsourcing humans to ask and answer questions about Wikipedia [16]; since the text of Wikipedia articles is under a Creative Commons license, this derived data set can be shared freely with few restrictions.

Use expressive deep-learning frameworks

The gradient-based optimization problems that arose in adopting deep-learning approaches in CV were originally solved by hand-coded routines for calculating loss functions and their gradients, and using either general-purpose or hand-coded optimization schemes. While good hand-coded implementations would perform sufficiently well in practice, they were often complicated to modify, making not-trivial changes to the model or to the training process difficult to implement.

Modern deep-learning frameworks are now built around a computation graph abstraction; the loss function being optimized is built up in a declarative fashion recursively from a library of fundamental functions, and the gradient is automatically calculated by the framework using automatic differentiation. Rather than using a general-purpose optimization routine, a machine learning (ML)-specific algorithm like AdaGrad [17] is often used for the subgradient optimization.

Using modern frameworks makes it much easier for a researcher or ML engineer to modify existing models and to mix and match different architectural choices, for example, to better explore the hyperparameter space of model structures. These frameworks also make it easier to transition code from research prototypes or exploratory models to production-ready code.

Share (at least some of) your code

The adoption of flexible frameworks makes it possible to share code with the understanding that it can be run by other researchers, and perhaps repurposed for other problems. Building off of the arXiv (<https://arxiv.org>), a repository of preprints that often is the debut location for new deep-learning techniques and applications, GitXiv (<http://www.gitxiv.com/>) links articles to GitHub code repositories of implementations of the

algorithms in the preprints, often authored by other people than the original authors of the articles.

The sharing of code based on common frameworks lends itself to combining components in new ways, much like interlocking plastic toy bricks. For example, many image captioning systems can be concisely described as simply plugging a convolutional neural network used for image representation into a recurrent neural network used for language modeling. 🎨

References

[1] Gatys LA, Ecker AS, Bethge M. “Image style transfer using convolutional neural networks.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; 2016 Jun 27–30; Las Vegas, NV; doi: 10.1109/CVPR.2016.265.

[2] Krizhevsky A, Sutskever I, Hinton GE. “ImageNet classification with deep convolutional neural networks.” In: Pereira F, Burgess CJC, Bottou L, Winberger KQ, editors. *Advances in Neural Information Processing Systems (NIPS 2012)*. Neural Information Processing Systems Foundation, Inc.; 2012.

[3] Simonyan K, Zisserman A. “Very deep convolutional networks for large-scale image recognition.” 2014. Cornell University Library. arXiv:1409.1556.

[4] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. “Going deeper with convolutions.” In: *2015 IEEE Conference on Computer Vision and Pattern Recognition*; 2015 Jun 7–12; Boston, MA. doi: 10.1109/CVPR.2015.7298594.

[5] He K, Zhang X, Ren S, Sun J. “Deep residual learning for image recognition.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; 2016 Jun 27–30; Las Vegas, NV. doi: 10.1109/CVPR.2016.90.

[6] Lowe D. “Distinctive image features from scale-invariant keypoints.” *International Journal of Computer Vision*. 2004;60(2):91–110. Available at: <https://doi.org/10.1023/B:VISI.0000029664.99615.94>.

[7] Bay H, Ess A, Tuytelaars T, Van Gool L. “Speeded-up robust features (SURF).” *Computer Vision and Image Understanding*. 2008;110(3):346–359. Available at: <https://doi.org/10.1016/j.cviu.2007.09.014>.

[8] Calonder M, Lepetit V, Strecha C, Fua P. “Brief: Binary robust independent elementary features.” In: *Computer Vision—ECCV 2010*; 2010 Sep 5–11; Crete, Greece: pp. 778–792.

[9] Rublee E, Rabaud V, Konolige K, Bradski G. “ORB: An efficient alternative to SIFT or SURF.” In: *Proceedings of the 2011 International Conference on Computer Vision*; 2011 Nov 6–13; Barcelona, Spain: pp. 2564–2571. doi:10.1109/ICCV.2011.6126544.

[10] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. “ImageNet: A large-scale hierarchical image database.” In: *IEEE Conference on Computer Vision and Pattern Recognition*; 2009 Jun 20–25; Miami, FL. doi: 10.1109/CVPR.2009.5206848.

[11] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems.” 2016. Cornell University Library. arXiv:1603.04467.

[12] Nilsback ME, Zisserman A. “A visual vocabulary for flower classification.” In: *2006 IEEE Conference on Computer Vision and Pattern Recognition*; 2006 Jun 17–23; New York, NY. doi: 10.1109/CVPR.2006.42.

[13] Welinder P, Branson S, Mita T, Wah C, Schroff F, Belongie S, Perona P. “Caltech-UCSD Birds 200.” *California Institute of Technology*. 2010. CNS-TR-2010-001.

[14] LeCun Y, Bottou L, Bengio Y, Haffner P. “Gradient-based learning applied to document recognition.” *Proceedings of the IEEE*. 1998;86(11):2278–2324. doi:10.1109/5.726791.

[15] Krizhevsky A, Hinton G. *Learning Multiple Layers of Features from Tiny Images*. 2009. Available at: <http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.

[16] Rajpurkar P, Zhang J, Lopyrev K, Liang P. “SQuAD: 100,000+ questions for machine comprehension of text.” 2016. Cornell University Library. arXiv:1606.05250.

[17] Duchi J, Hazan E, Singer Y. “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of Machine Learning Research*. 2011;12:2121–2159. Available at: <https://dl.acm.org/citation.cfm?id=2021068>.



DEEP LEARNING FOR SCIENTIFIC DISCOVERY

by Courtney D. Corley, Nathan O. Hodas, Enoch Yeung, Alexander Tartakovsky, Tobias Hagge, Sutanay Choudhury, Khushbu Agarwal, Charles Siegel, and Jeff Daily

"If a typical person can do a mental task with less than one second of thought, we can probably automate it using AI [artificial intelligence] either now or in the near future."

Andrew Ng, former Chief Scientist of Baidu

Deep learning has positively impacted fields in which perceptive tasks are paramount, such as computer vision and natural language processing. Tremendous strides have also been made in deep reinforcement learning towards robotics and gaming. Given the success in these fields, it is apparent that deep learning may also accelerate scientific discovery. Herein, we discuss several topic areas in which modern representation learning is driving innovation. This article describes learning nonlinear models from data, learning scientific laws from models, deep learning on graphs, and approaches to scalable deep learning.

Data-driven discovery and deep Koopman operators

Scientists possess powerful tools to deal with linear models, where the output of the system is directly proportional to its inputs. But, the world is not linear. Yet for as long as science has existed, we have had limited capabilities to probe and understand nonlinear models. Huge swaths of reality remain inaccessible. In 1931, B. O. Koopman proposed the existence of a mathematical function that turned a nonlinear system into an infinite dimensional linear system [1], a function we now call the Koopman operator. He did not show how to find it; for generations, the Koopman operator proved to be too difficult to compute.

Recently, advanced computational methods such as dynamic mode decomposition (DMD) [2, 3, 4] have renewed interest in finding the Koopman operator [5–7]. DMD requires the scientist to choose certain dictionary functions, based on their experience or imagination; a poor choice results in an inaccurate Koopman operator. New methods, based on deep learning [8] and shallow learning [9], provide a way to automatically update dictionaries during training. While previous methods rely on static dictionaries, this new approach allows for dynamic editing and real-time updating of dictionaries during the training process—thus finding a high-fidelity Koopman operator as the dictionaries are refined (see figure 1).

Deep Koopman operators have the potential to be transformational in data-driven scientific discovery in applications ranging from synthetic biology to control theory.

Model-driven discovery in deep learning

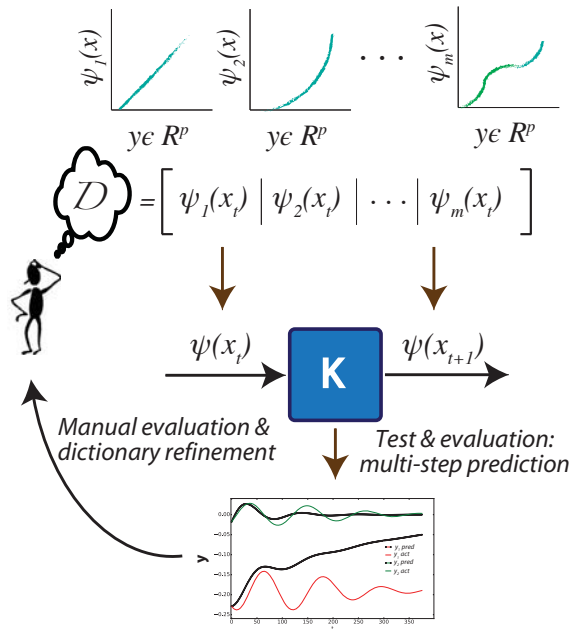
Scientific computing is all about predictions—predicting properties of materials, behavior of systems, etc. From a physics point of view, predictions are possible because all natural and engineered systems follow conservation laws. Traditionally, parameters and rates in these laws are estimated from data, rigorous mathematical techniques, and/or expert knowledge. Neural networks are trained on data, which can only come from the past and present. Therefore, making accurate predictions of systems using deep learning in the absence of tight coupling with conservation laws depends on the ability of a neural network itself to learn conservation laws. At present, this remains an open question.

Conservation laws manifest themselves as a set of algebraic or dynamical constraints. There are two direct methods that can be used to incorporate conservation laws into neural networks. One direct method is to use deep learning as a part of a solution of conservation equations to directly incorporate conservation laws into neural networks [10]. Another direct method is to develop dedicated modules within a larger network that approximate the system response defined by the scientific law. For example, when describing local behavior of a nonlinear system, it is possible to use the layers of a convolutional neural network (CNN) to directly encode the system dynamic response as a part of the network.

Another approach to enforce scientific conservation laws is to modify the objective function for training. The unconstrained objective function for a deep neural network restricts the input-output map of the deep neural network to match the input-output relationships observed in the data. However, one can impose conservation laws as a constraint on the optimization problem. Since constrained optimization problems can be formulated using objective functions with Lagrangian multipliers, simply modifying the objective function to incorporate a conservation law, $g(x) = 0$, enforces learning of the conservation law during training.

CURRENT METHOD

Extended Dynamic Mode Decomposition



OUR APPROACH

Deep Dynamic Mode Decomposition

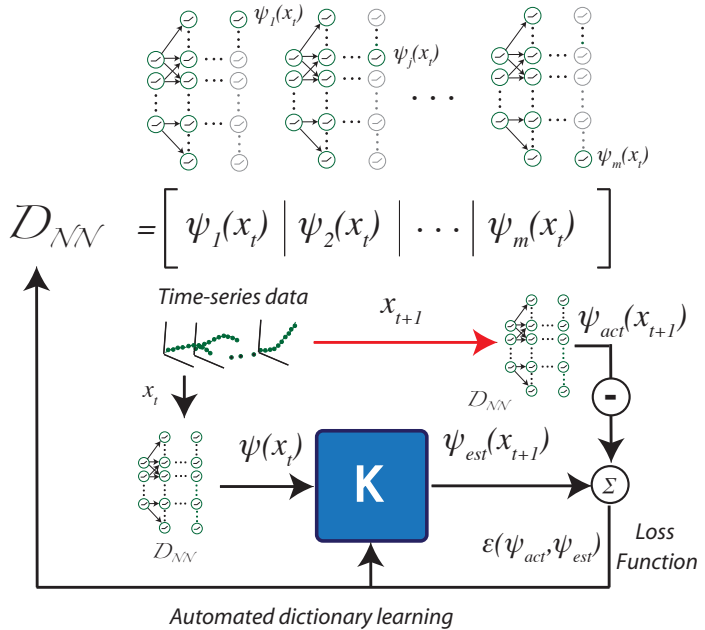


FIGURE 1. This schematic illustrates the state-of-the-art approach versus a deep learning approach for Koopman operator learning. In existing methods, the scientist postulates a variety of reasonable dictionary functions to include in their analysis (e.g., Hermite or Legendre polynomials, thin-plate radial basis functions). If the model is not predictive, the burden is on the scientist to postulate new functions to include in the dictionary. In our approach, the entire lifting map is specified as the output or outputs of a deep artificial neural network. The deep neural network can be a multilayer feed-forward network, a convolutional network, a recurrent network, etc. The prediction error is then used to compute the loss function and refine the dictionary automatically. Regardless of the structure, the network learns the most efficient dictionary for the given system.

Deep learning for graphs

One other vital area in scientific discovery in which deep learning can make an impact is the ability to reason about and learn over graphs. Introduced by Leonhard Euler in 1736, graphs have emerged as a critical tool for modeling a large number of real-world data sources we deal with today. These include databases containing relationships between many entities such as computer or social networks as well as semantic networks involving entities that are instantiations of different concepts (e.g., people, locations, organizations), and the relationships or interactions between these entities. All prominent graph algorithms seek to extract different types of information from this web of relationships, such as connecting entities (graph search), grouping entities with similar behavior (clustering), predicting the relationship

between two entities (link recommendation or graph completion), or finding entities that interact closely (community detection).

Application of machine learning (ML) techniques into graph-oriented tasks typically begins with learning a set of features for every entity (or node) in the graph. We translate a graph into a tabular representation where the features learned on every node preserve the topological properties associated with its neighborhood. This idea was first proposed by Scarselli et al. [11] and more recently improved upon by DeepWalk [12] and node2vec [13]. These represent an innovation in representation learning, where task-independent features are learned in a data-driven fashion as opposed to manual feature engineering, and where their algorithmic accuracy closely matches task-specific approaches.

Recently, a wave of research has demonstrated the effectiveness of deep learning-based approaches for graph-based tasks. Algorithms that traditionally relied on graph walks are naturally adopted towards recurrent neural networks and its variants for tasks such as knowledge base completion and probabilistic reasoning [14], shortest-path queries, and deductive reasoning [15]. CNNs are a natural choice for algorithms that treat graphs as matrices [16] or where operations on a node and its neighborhood are key [17]. Combining these individual building blocks to reflect the hierarchical structure that naturally exists in data is another emerging area of research [18]. The potential to leverage graph-based deep learning is only now being realized for scientific discovery.


Scalable deep-learning algorithms on extreme-scale architectures

Even when an accurate mathematical description of a system of interest exists, numerically solving the resulting equations can be difficult. This is especially true when results are needed in real time. Deep neural networks are difficult to train, so researchers are increasingly turning to high-performance computing (HPC) resources to scale their deep learning to match the complexity of their problems. It is also important to connect research solutions and users who would like to use the systems without needing detailed knowledge of the system architectures.

Recent developments, such as the Machine Learning Toolkit for Extreme Scale (MaTeX), are making progress by enabling a slew of algorithms to scale ML algorithms on HPC systems by extending

publicly available single-node implementations of major toolkits such as Caffe, PyTorch, TensorFlow, and Keras¹. The developments include: a) scalable implementations that use high-performance communication libraries, b) scalable implementations to overlap communication with computation using layer-wise gradient descent, c) techniques to adaptively prune deep neural network (DNN) topologies on the fly, d) methods to adaptively grow the DNNs from blueprints, and e) adaptively remove data samples that are unlikely to contribute to model learning. These tools encapsulate approaches to abstract the changes for distributed memory from the users leveraging Google TensorFlow and Keras.

Summary

Deep learning in the pursuit of scientific discovery is showing great progress and promise for increasing the predictive power and efficiency of computational and data-driven models. A benefit of deep learning is that humans are not needed for programming explicit rules. Deep neural networks learn on their own, building representations and gathering information from larger data sets, searching for patterns and anomalies that humans wouldn't normally find or know how to process. There have been unprecedented investments by industry, academia, and government, and many challenges remain in the applicability of deep learning outside of perception tasks. While a cautious approach should be considered when leveraging these techniques, we are optimistic that progress can and will be made in the pursuit of scientific discovery. 

References

[1] Koopman BO. "Hamiltonian systems and transformations in Hilbert space." *Proceedings of the National Academy of Sciences of the USA*. 1931;17(5):315–318. Available at: <http://www.pnas.org/content/17/5/315>.

[2] Mezić I. 2005. "Spectral properties of dynamical systems, model reduction and decompositions." *Nonlinear Dynamics*. 2005;41(1–3):309–325. Available at: <https://doi.org/10.1007/s11071-005-2824-x>.

[3] Williams MO, Kevrekidis IG, Rowley CW. "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition." *Journal of Nonlinear Science*. 2015;25(6):1307–1346.

[4] Proctor JL, Brunton SL, Kutz JN. "Dynamic mode decomposition with control." *SIAM Journal on Applied Dynamical Systems*. 2016;15(1):142–161. Available at: <https://doi.org/10.1137/15M1013857>.

1. See <https://github.com/matex-org/matex/wiki>.

- [5] Rowley CW, Mezić I, Bagheri S, Schlatter P, Henningson, DS. 2009. "Spectral analysis of nonlinear flows." *Journal of Fluid Mechanics*. 2009;641:115–127. Available at: <http://doi.org/10.1017/S0022112009992059>.
- [6] Susuki Y, Mezić I. 2014. "Nonlinear Koopman modes and power system stability assessment without models." *IEEE Transactions on Power Systems*. 2014;29(2):899–907. doi:10.1109/TPWRS.2013.2287235.
- [7] Mauroy A, Mezić I. "Global stability analysis using the eigenfunctions of the Koopman operator." *IEEE Transactions on Automatic Control*; 2016 61(11):3356–3369. doi: 10.1109/TAC.2016.2518918.
- [8] Hodas N, Kundu S, Yeung E. "Learning deep neural network representations for Koopman operators of nonlinear dynamical systems." 2017. Cornell University Library. arXiv:1708.06850
- [9] Li Q, Dietrich F, Bollt EM, Kevrekidis I G. "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator." 2017. Cornell University Library. arXiv:1707.00225.
- [10] Hagge T, Stinis P, Yeung E, Tartakovsky AM. "Solving differential equations with unknown constituent relations as recurrent neural networks." Cornell University Library. arxiv:1710.02242.
- [11] Scarselli F, Marco G, Tsoi AC. "The graph neural network model." *IEEE Transactions on Neural Networks*. 2009;20(1):61–80. doi: 10.1109/TNN.2008.2005605.
- [12] Perozzi B, Al-Rfou R, Skiena S. "Deepwalk: Online learning of social representations." In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; 2014 Aug 24–27; New York, NY: doi: 10.1145/2623330.2623732.
- [13] Grover A, Leskovec. "node2vec: Scalable feature learning for networks." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; 2016 Aug 13–17; San Francisco, CA: doi: 10.1145/2939672.
- [14] Graves A, Wayne G, Reynolds M, Harley T, Danihelka I, Grabska-Barwinska A, Comenarejo SG, Ramalho T, Agapiou J, Badia AP et al. "Hybrid computing using a neural network with dynamic external memory." *Nature International Journal of Science*. 2016;538:471–476. doi: 10.1038/nature20101.
- [15] Zhiling L, Liu L, Yin J, Ying L, Wu Z. "Deep learning of graphs with Ngram convolutional neural networks." *2017 IEEE Transactions on Knowledge and Data Engineering*. 2017;29(10):2125–2139. doi: 10.1109/TKDE.2017.2720734.
- [16] Duvenaud D, Maclaurin D, Aguilera-Iparraguirre JA, Gomez-Bombarelli RG, Hirzel T, Aspuru-Guzik A, Adams RP. "Convolutional networks on graphs for learning molecular fingerprints." 2015. Cornell University Library. arXiv:1509.09292.
- [17] Yang Z, Yang D, Dyer C, He X, Smola AJ, Hovy EH. "Hierarchical attention networks for document classification." In: *Proceedings of NAACL-HLT*. 2016 Jun 12–17.
- [18] Niepert M, Ahmed M, Kutzkov K. "Learning convolutional neural networks for graphs." In: *Proceedings of the 33rd International Conference on Machine Learning*. New York (NY): JMLR W&CP;2016. Available at: <http://proceedings.mlr.press/v48/niepert16.pdf>.

Extremely rare phenomena and sawtooths in La Jolla

by Anthony C. Gamst and Skip Garibaldi

Machine learning (ML) problems faced by the NSA and Department of Defense (DoD) are different from those found in industry and academia. For example, one often wants to develop classifiers with respectable true-positive rates at extremely low false-positive rates, far lower than what is routinely considered in other contexts. Inspired by such challenges, ML research at the Institute for Defense Analyses' Center for Communications Research in La Jolla, California has focused on adapting commercial and academic techniques to this more austere regime as well as the theoretical and foundational questions raised by that adaptation.

How our problems are different

The ML challenges faced by the NSA, and more generally by the DoD, are somewhat different from those faced by academia or industry. For example, the NSA often wants to detect extremely rare phenomena, in the sense that, of 10 million observations, only a few will be true targets. In that situation, one has to aim for an extremely low false-positive rate to ensure that the number of observations that need to be checked in greater detail (for example, by an analyst) is not immense. The ML research we have done in La Jolla has focused on addressing ML in that domain, and this has in turn spurred more research on the foundations of ML.

Achieving an extremely low false-positive rate

Many immediate practical issues arise in building a detector for extremely rare phenomena. Here is a trivial one: A popular way to present the efficacy of a detector is by its receiver operating characteristic (ROC) curve, which is a plot with the false-positive rate (FPR, the proportion of negative observations that were marked as positive) on the horizontal axis and the true-positive rate (TPR, the proportion of positive observations that were marked as positive) on the vertical axis, where both axes run from 0 to 1.

When targeting extremely rare phenomena, one is interested in the TPR when the FPR is, say, 10^{-6} , and comparing that with the TPR when the FPR is 10^{-7} . Such distinctions are invisible on a normal ROC curve. This illustrates how our problem regime is different from the usual problem regime one reads about in ML publications; it also demonstrates that standard characterizations of diagnostic accuracy, such as area under the ROC curve, are too coarse to capture the differences of interest to us. Of course, plotting the ROC curve on a semi-log scale, where the horizontal axis is $\log(\text{FPR})$ as in figure 1, improves the visibility of the low-FPR end of the ROC curve, and this helps with analysis.

A more serious issue stems from the relative proportions of the positive and negative items in the training data. To train a detector, it is not reasonable to use data occurring at the same frequencies as it does in nature. Indeed, the power of a classifier

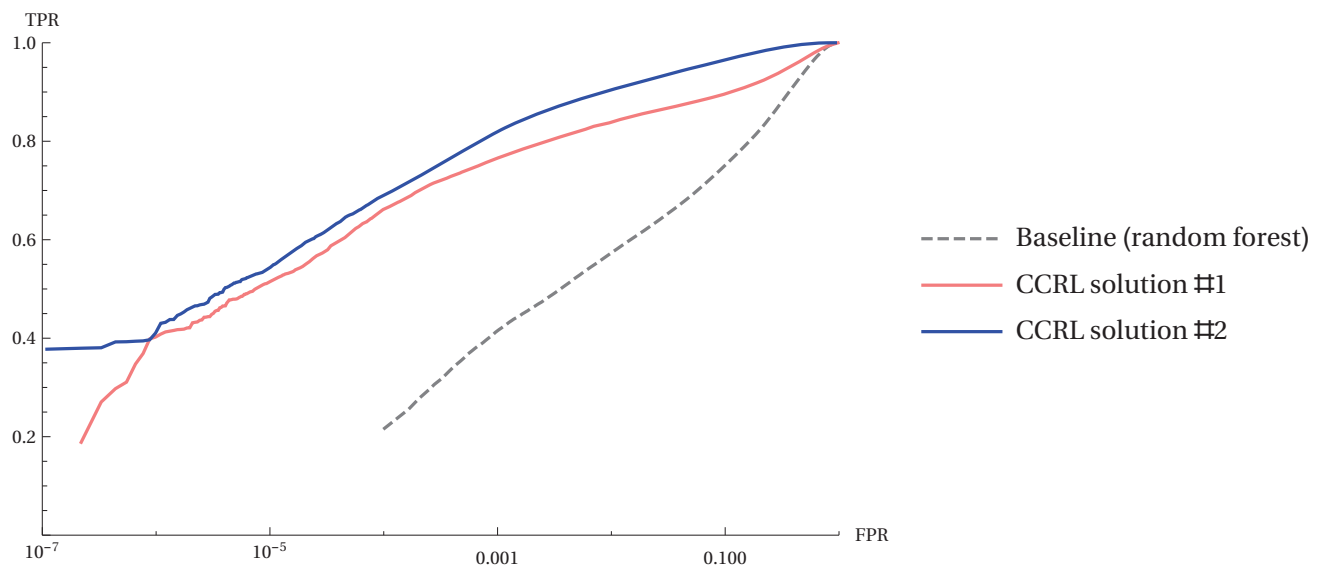


FIGURE 1. This semi-log plot shows the TPR and log FPR achieved by a detector for various choices of threshold. The goal is to get the left-hand endpoint of the ROC curve as close as possible to the upper left corner.

is determined almost entirely by the proportion of the training set made up of the smallest class, so one needs approximately equal amounts (or at least similar orders of magnitude) of both positive and negative observations. Typically, one needs to synthesize data for this. Of course, the goal of ML is to characterize a particular data-generating process, and synthetic data might be generated in a manner that differs in important but unknown ways from the process of interest. As a result, using synthetic data to train a neural network, for example, might lead to a system with excellent apparent accuracy in the lab but poor performance in practice. Furthermore, in order to feel any confidence at all about a detector's performance at an FPR of one-in-a-million, one needs millions of negative examples in the test set and therefore a corresponding number of positive examples.

We generated training data for one practical problem in which the users required an FPR of 10^{-6} and hoped for a TPR of 0.1 at that FPR. (Note the contrast to more familiar settings, where the target FPR and TPR are both larger.) When we got our hands on the problem, the users had already trained a classifier using a random forest package. They thought their classifier might be sufficient but weren't sure because they had not yet accumulated enough test data. We tried a variety of ML techniques, ranging from off-the-shelf

to invented in-house, and found that a small ensemble of modestly sized neural networks performed best (see figure 1). These networks are indeed quite modest: Each had eight layers and about 1,800 nodes, which is already tiny compared with AlexNet (small by current standards but state of the art in 2012) with 650,000 nodes [1]. An ensemble of these modest networks was produced by averaging their outputs rather than applying more sophisticated methods such as bagging, boosting, or arcing as described in chapter 16 of [2]. Figure 1 shows the performance goals were handily met, while the baseline random forest approach was unable to even achieve the 10^{-6} FPR. The bound in performance by the forest arose because the forest scored at least 1-in- 10^4 negative items at least as high as the highest-scoring positive item.

One typical difficulty, which also appeared in this problem, is that some of the features we wanted to use for prediction had very poor statistical properties. This is often addressed by examining a histogram or summary statistics for each feature and deciding what mitigations are warranted. (Google Facets [3] is one well-publicized tool for this task.) However, this data set had more than 2,200 features, making feature-by-feature inspection tiresome, so we employed automated techniques to cull features that were constant or mostly constant and features that had many

missing values. More interestingly, many features had very large skewness or kurtosis, which would not be mitigated by the usual “z-scoring” normalization procedure in which one subtracts the mean and divides by the standard deviation. We addressed this deficiency by scaling and shifting the values so that the 25th percentile and 75th percentile values were set to -1 and 1 respectively, and we then clipped the values to lie between -10 and 10. Using this normalization procedure outperformed both the naive z-scoring procedure and a modification of the Box-Cox transformation from [4]. Note that these procedures can be applied in an automated fashion and the parameters of the Winsorization—the clipping at -10 and +10—are learned from the data.

Do you believe in ROC curves? Should you?

In the discussion of ROC curves for extremely low FPR, it is natural to wonder how much one should trust the TPR asserted at an FPR of, say, 10^{-7} . At the very least, one would like to be able to determine whether or not the ROC curve corresponding to one technique is significantly different from that of another technique, either over some range of FPRs or at a specific FPR.

The paper [5] discusses bootstrap methods for constructing confidence bounds for ROC curves associated with ensemble methods, like random forests, and uses those methods to study the accuracy of random forests in a particular prediction problem in which the desired FPR was one-in-a-million, as in the preceding example. One advantage of the techniques discussed in [5] is that, when the ensemble methods base their decisions on votes made by individual weak classifiers, the confidence bounds can be constructed without resampling the training data or refitting the models, producing significant computational savings.

Using these techniques, Gamst, Reyes, and Walker [5] were able to determine the effect of increasing the number of trees in the forest, increasing or decreasing the amount of data used to fit the individual classification trees, increasing or decreasing the number of features considered at each node of the individual trees, selecting the depth of the individual trees, and modifying the voting or splitting rules used by the forest. These techniques were also used in a large-scale

study of various ML techniques—some new and some off the shelf—applied to the same classification problem, allowing us to determine whether any of the models were significantly more accurate than any of the others and which of the models made the most accurate predictions.

Capacity of neural networks

Another direction of unclassified research in La Jolla has concerned the expressiveness of neural networks. It is well known as a theoretical result that sufficiently large neural networks can represent essentially any function. However, in practice, the range of functions that can be approximated by any particular neural network is less clear. For example, consider a neural network with one input and one output, consisting of D dense layers, each with W nodes and the commonly used ReLU activation function. What functions can such a network learn? Clearly the output is a linear spline. How many knots can the spline have? Provably the answer is $O(W^D)$ [6], but the average network exhibits many fewer knots [7]. In fact, randomly initialized networks almost always produce a function with $O(WD)$ knots.

These results describe a neural network just after initialization, before training, and they show that the untrained neural network tends to represent a function that is far less complex than what is possible in the space of functions representable by the network. (In fact, even after training, such networks tend to produce functions with $O(WD)$ knots, but the placement of the knots is adapted to the underlying function the network is trying to learn; see [8].)

What about the complexity of functions represented by trained networks? Two La Jolla researchers [9] handcrafted a network to represent a function whose graph is a hump adorned with a sawtooth; the output of this network is the blue plot in figure 2. They then took a network with an identical architecture, initialized its weights randomly (as is common practice), and trained it to model the same function. The plot of the output of the trained network is the green plot in figure 2, and clearly does not precisely match the output of the crafted network. This is either a plus (if you view the sawtooth as noise in your true measurement of the hump) or a minus (if you think the sawtooth is part of the signal). The experiment was repeated

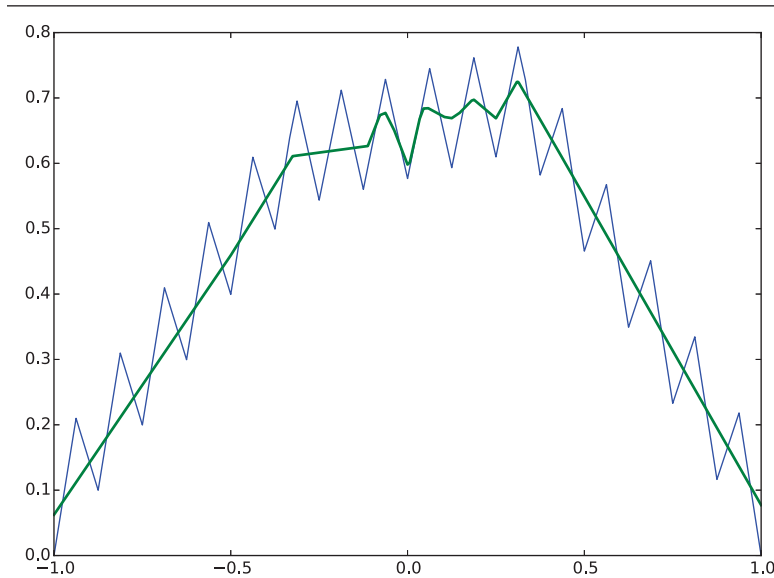


FIGURE 2. Input versus output of two neural networks. The blue sawtooth is the output of a network with handcrafted weights. The green curve is the output of a trained network, the best fit among 50,000 networks of identical architecture trained on the output of the handcrafted network.

50,000 times, and only the best fit is reported in the figure, so it is clear that even the best fit from the network has much lower complexity than the underlying function, in spite of the fact that there is a set of weights that leads this network to reproduce the bent sawtooth function exactly. The point is that a trained network does not fully explore the space of functions that the network could produce.

This last result sheds light on two different threads of outside research while not belonging to either. The first thread is a series of papers giving lower bounds on the complexity of a neural network that can approximate a given function well; see, for example, [10], [11], [12], or [13]. These results are about networks with handcrafted weights and do not attempt to give lower bounds on the complexity of a *trained* neural network that can approximate a given function. Indeed, the experiment described in the preceding paragraph shows that the two questions are different—trained neural networks are less expressive. So, while it is possible to construct networks capable of learning wavelet-like basis functions on smooth manifolds and carry the corresponding approximation-theoretic results from the wavelet domain to the neural network domain, it is unclear that such networks can actually be trained to represent the functions of interest.

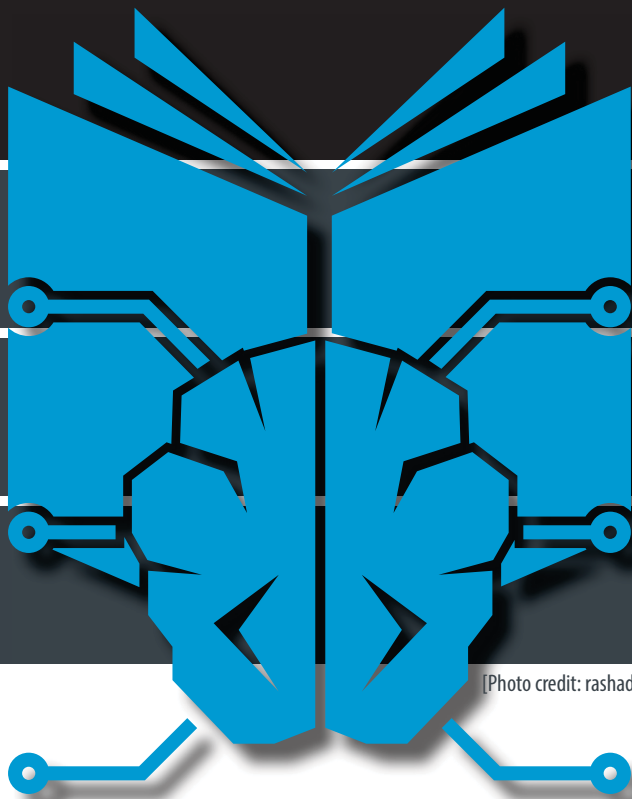
The second thread of research demonstrates that, under some hypotheses, the local optima found when training a neural network are “close to” global optima; see papers such as [14] and [15]. These papers usually deal with somewhat unrealistic idealizations of networks and, as [14] says: “[do] not yet directly apply to the practical situation.” The results of [8], on the other hand, demonstrate that it is easy to construct networks and learning problems for which local optima are far from global optima. As in [15], the distance between local and global optima is a function of the size of the network, with larger networks being more expressive and having a greater capacity for overfitting. A natural question is: How large a network do you need for the problem you actually have (and how can you tell when your network is too large or too small)? Of course, the answer depends on the complexity of the function you are trying to learn and how much data you have. Ongoing research (including [8]) attempts to produce these and other diagnostics. 📺

References

- [1] Krizhevsky A, Sutskever I, Hinton GE. “ImageNet classification with deep convolutional neural networks.” In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*. 2012. pp. 1097–1105.
- [2] Hastie T, Tibshirani R, Friedman J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer; 2009. ISBN: 978-1-4899-0579-2.
- [3] Google Facets. Available at: <http://github.com/PAIR-code/facets>.
- [4] Box GEP, Cox DR. “An analysis of transformations.” *Journal of the Royal Statistical Society B*. 1964;26(1):211–252.
- [5] Gamst AC, Reyes JC, Walker A. “Estimating the operating characteristics of ensemble methods.” 2017. Cornell University Library. arXiv:1710.08952.
- [6] Chen KK. “The upper bound on knots in neural networks.” 2016. Cornell University Library. arXiv:1611.09448.

- [7] Chen KK, Gamst AC, Walker A. “Knots in random neural networks.” Presented at the Thirtieth Annual Conference on Neural Information Processing Systems (NIPS 2016), Workshop on Bayesian Deep Learning; 2016 Dec 5–10; Barcelona, Spain. Available at: http://bayesian-deeplearning.org/papers/BDL_2.pdf.
- [8] Chen KK, Gamst AC, Walker A. “The empirical size of trained neural networks.” 2016. Cornell University Library, arXiv:1611.09444.
- [9] Gamst AC, Walker A. “The energy landscape of a simple neural network.” 2017. Cornell University Library. arXiv:1706.07101.
- [10] Barron A. “Universal approximation bounds for superpositions of a sigmoidal function.” *IEEE Transactions on Information Theory*. 1993;39(3):930–945. doi: 10.1109/18.256500.
- [11] Bolcskei H, Grohs P, Kutyniok G, Petersen P. “Optimal approximation with sparsely connected deep neural networks.” 2017. Cornell University Library. arXiv:1705.01714, 2017.
- [12] Schmitt M. “Lower bounds on the complexity of approximating continuous functions by sigmoidal neural networks.” In: Solla SA, Leen TK, Muller K-R, editors. *Advances in Neural Information Processing Systems 12 (NIPS 1999)*. The MIT Press; 2000. pp. 328–334.
- [13] Shaham U, Cloninger A, Coifman RR. “Provable approximation properties for deep neural networks.” *Applied and Computational Harmonic Analysis* (in press). Available at: <https://doi.org/10.1016/j.acha.2016.04.003>.
- [14] Kawaguchi K. “Deep learning without poor local minima.” In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R, editors. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*; 2016.
- [15] Choromanska A, Henaff M, Mathieu M, Arous GB, LeCun Y. “The loss surfaces of multilayer networks.” In: *Proceedings of Machine Learning Research: Artificial Intelligence and Statistics*, 9–12 May 2015, San Diego, CA. 2015;38:192–204.

AT A GLANCE

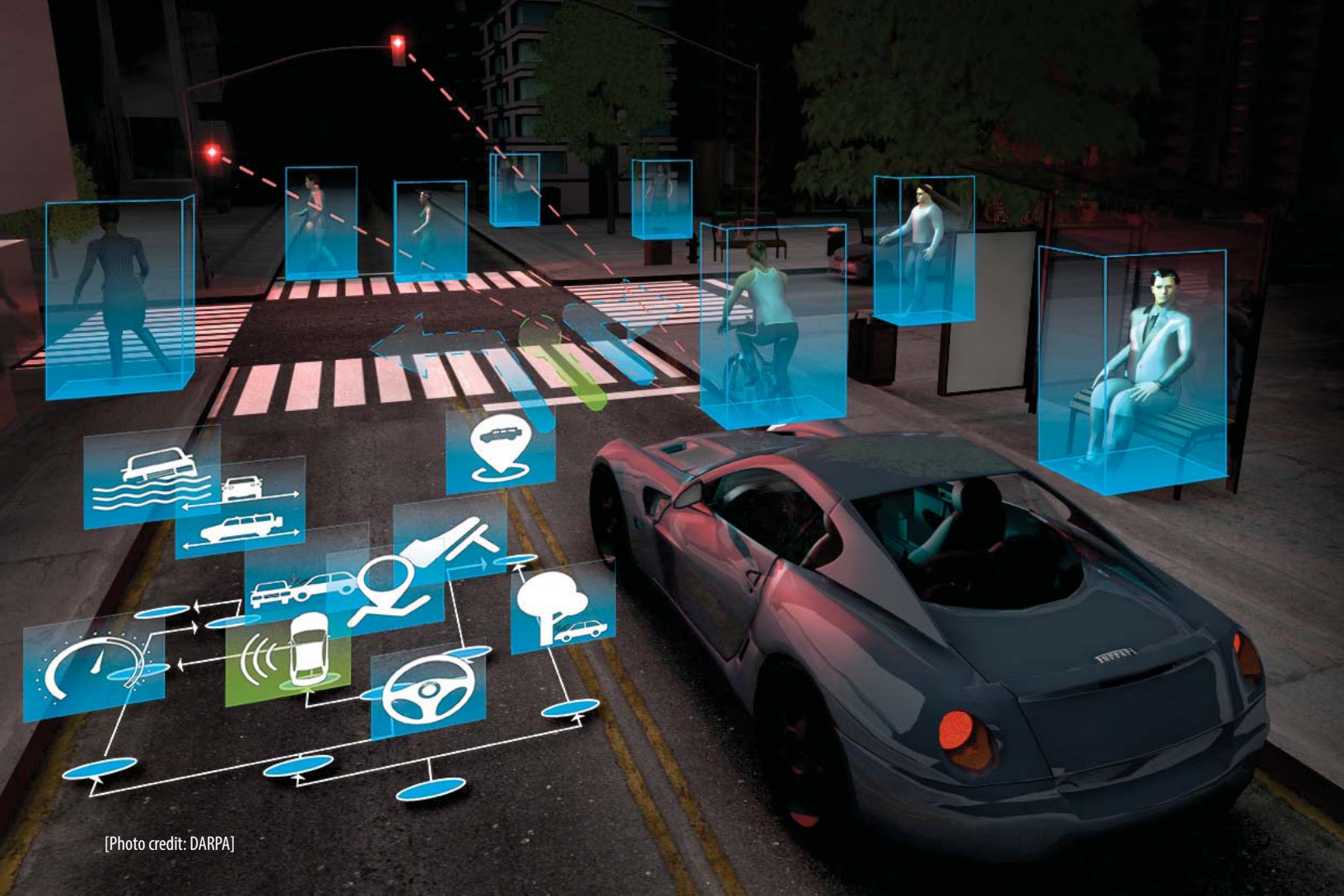


MACHINE LEARNING PROGRAMS ACROSS THE GOVERNMENT

[Photo credit: rashadashurov/iStock/Thinkstock]

IARPA Machine Intelligence from Cortical Networks program *Program Manager: David Markowitz*

The Intelligence Advanced Research Projects Activity (IARPA) Machine Intelligence from Cortical Networks (MICrONS) program aims to achieve a quantum leap in machine learning by creating novel machine learning algorithms that use neurally inspired architectures and mathematical abstractions of the representations, transformations, and learning rules employed by the brain. To guide the construction of these algorithms, researchers will conduct targeted neuroscience experiments that interrogate the operation of mesoscale cortical computing circuits, taking advantage of emerging tools for high-resolution structural and functional brain mapping. The program is designed to facilitate iterative refinement of algorithms based on a combination of practical, theoretical, and experimental outcomes. The researchers will use their experiences with the algorithms' design and performance to reveal gaps in their understanding of cortical computation and will collect specific neuroscience data to inform new algorithmic implementations that address these limitations. Ultimately, as the researchers incorporate these insights into successive versions of the machine learning algorithms, they will devise solutions that can achieve human-like performance on complex information processing tasks with human-like proficiency. For more information on this program, visit <https://www.iarpa.gov/index.php/research-programs/microns>.



[Photo credit: DARPA]

DARPA Lifelong Learning Machines program

Program Manager: Hava Siegelmann

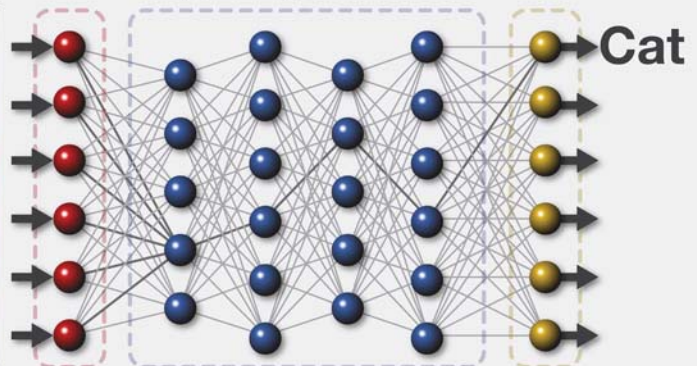
The Defense Advanced Research Projects Agency (DARPA) Lifelong Learning Machines (L2M) program considers inspiration from biological adaptive mechanisms as a supporting pillar of the project. Biological systems exhibit an impressive capacity to learn and adapt their structure and function throughout their life span, while retaining stability of core functions. Taking advantage of adaptive mechanisms evolved through billions of years honing highly robust tissue-mediated computation will provide unique insights for building L2M solutions. The first technical area of the L2M program will focus on functional system development and take inspiration from known biological properties. The second technical area will involve computational neuroscientists and computational biologists in identifying and exploring biological mechanisms that underlie real-time adaptation for translation into novel algorithms. These will possibly lead to the development of a plastic nodal network (PNN)—as opposed to a fixed, homogeneous neural network. While plastic, the PNN must incorporate hard rules governing its operation, maintaining an equilibrium. If rules hold the PNN too strongly, it will not be plastic enough to learn, yet without some structure the PNN will not be able to operate at all. For more information on this program, visit <https://youtu.be/JeXv48AXLbo>.

DARPA Explainable Artificial Intelligence program

Program Manager: Dave Dunning

The DARPA Explainable Artificial Intelligence (XAI) program goal is to create a suite of machine learning techniques that can produce more explainable models while maintaining a high level of learning performance (i.e., prediction accuracy), and enable human users to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners. The program will focus the development of multiple systems on addressing challenge problems in two areas: 1) machine learning problems to classify events of interest in heterogeneous, multimedia data and 2) machine learning problems to construct decision policies for an autonomous system to perform a variety of simulated missions. These two challenge problem areas were chosen to represent the intersection of two important machine learning approaches (i.e., classification and reinforcement learning) and two important operational problem areas for the Department of Defense (i.e., intelligence analysis and autonomous systems). For more information on this program, visit <https://www.darpa.mil/program/explainable-artificial-intelligence>.

Machine Learning System



This is a cat.

Current Explanation

This is a cat:

- It has fur, whiskers, and claws.
- It has this feature:



XAI Explanation

AlgorithmHub provides robust environment for NSA machine learning research

Being able to save time, money, and effort while achieving mission can be easily classified as a “win-win situation” for everyone involved. Fortunately, researchers at NSA Hawaii (NSAH) are doing just that to further develop their machine learning (ML) models.


Teams at NSAH are conducting research on the training of ML models to provide a topical representation of foreign language content. The goal is to enable any user to quickly assign a meaning and perspective to text without having any prior knowledge of the language in which it is written. Their work will further develop program applications that will generate word lists and eventually visual images of data organized by topic. In need of a better way to continue their research, NSAH discovered Hawaii-based start-up AlgorithmHub at a speed networking event and quickly saw potential in what the company had to offer.

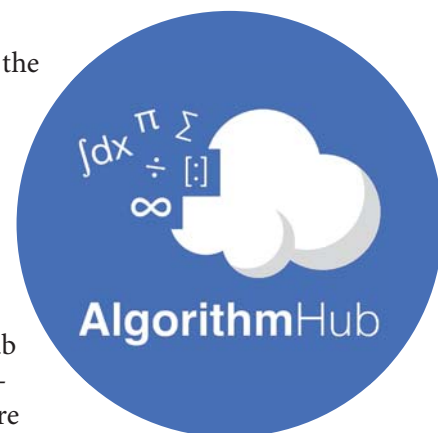
To meet NSAH requirements, the NSA Technology Transfer Program (TTP) finalized a Cooperative Research and Development Agreement (CRADA) with AlgorithmHub to apply their data science cloud compute environment to NSA’s unclassified ML research problems. The partnership with AlgorithmHub allows NSA researchers to deploy algorithms in a cloud environment without lengthy delays and costs associated with provisioning and maintaining virtual machines. AlgorithmHub’s unique Integrated Development Environment (IDE) for commercial cloud services provides data scientists with the ability to experiment and share their data science algorithms while leveraging the compute power and storage capacity of the cloud. With NSA participating as a beta test partner, the partnership will help boost the company profile for AlgorithmHub and allows them to test and determine the current limitations of their cloud capability.

After a highly successful data science workshop with participants from NSA, the Lincoln

Laboratory (managed by the Massachusetts Institute of Technology), and the Institute of Defense Analysis, NSA hosted a topic-modeling cohort, subsequently extending testing and evaluation time in the AlgorithmHub environment. An upcoming workshop will be more comprehensive and include broader participation to further refine the model for additional use of the AlgorithmHub platform for data analytics by NSAH. This effort is developing a potential model for continuing collaboration for data analytics and tradecraft development for use across the enterprise and with other researchers in academia and industry.

John Bay, CEO of MathNimbus Inc., DBA AlgorithmHub, said “Through our CRADA with a data science team at the NSA, we have enhanced efficiency and effectiveness in evaluating machine learning algorithms for topic identification. Moreover, based on features desired by the NSA data scientists, we implemented support for large-scale hyperparameter optimization and a collaborative Jupyter notebook environment in our AlgorithmHub software platform. These new features are not only valued by data scientists at the NSA, but also with other AlgorithmHub customers. The CRADA with the NSA has provided us critical feedback and validation needed to continue to evolve the AlgorithmHub platform into an innovative, commercially viable product.”

The NSA TTP, located within the Research Directorate, establishes partnerships with industry, academia, and other government agencies to help accelerate mission goals, advance science, foster innovation, and promote technology commercialization. 





NATIONAL SECURITY AGENCY



CENTRAL SECURITY SERVICE

Defending Our Nation. Securing The Future