O'REILLY®

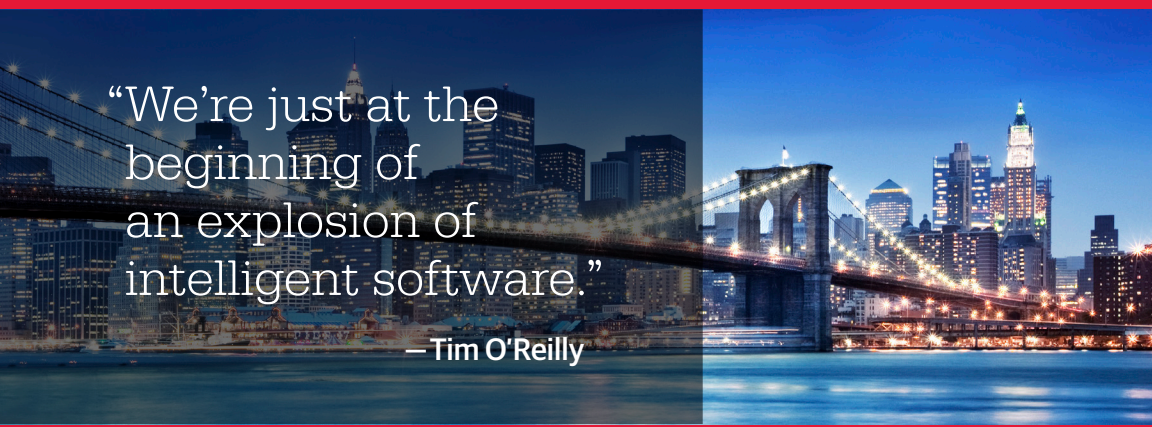
The Future of Machine Intelligence

Perspectives from Leading Practitioners

David Beyer

O'REILLY®

Artificial Intelligence



Conference Leadership

Tim O'Reilly
O'Reilly Media, Founder and CEO

Peter Norvig
Director of Research at Google Inc.

Explore opportunities for applied AI

oreillyaicon.com

The Future of Machine Intelligence

Perspectives from Leading Practitioners

David Beyer

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Future of Machine Intelligence
by David Beyer

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Shannon Cutt
Production Editor: Nicole Shelby
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

February 2016: First Edition

Revision History for the First Edition
2016-02-29: First Release

978-1-491-93230-8

[LSI]

Table of Contents

Introduction

Machine intelligence has been the subject of both exuberance and skepticism for decades. The promise of thinking, reasoning machines appeals to the human imagination, and more recently, the corporate budget. Beginning in the 1950s, Marvin Minksy, John McCarthy and other key pioneers in the field set the stage for today's breakthroughs in theory, as well as practice. Peeking behind the equations and code that animate these peculiar machines, we find ourselves facing questions about the very nature of thought and knowledge. The mathematical and technical virtuosity of achievements in this field evoke the qualities that make us human: Everything from intuition and attention to planning and memory. As progress in the field accelerates, such questions only gain urgency.

Heading into 2016, the world of machine intelligence has been bustling with seemingly back-to-back developments. Google released its machine learning library, TensorFlow, to the public. Shortly thereafter, Microsoft followed suit with CNTK, its deep learning framework. Silicon Valley luminaries recently pledged up to one billion dollars towards the OpenAI institute, and Google developed software that bested Europe's Go champion. These headlines and achievements, however, only tell a part of the story. For the rest, we should turn to the practitioners themselves. In the interviews that follow, we set out to give readers a view to the ideas and challenges that motivate this progress.

We kick off the series with Anima Anandkumar's discussion of tensors and their application to machine learning problems in high-dimensional space and non-convex optimization. Afterwards, Yoshua Bengio delves into the intersection of Natural Language Pro-

cessing and deep learning, as well as unsupervised learning and reasoning. Brendan Frey talks about the application of deep learning to genomic medicine, using models that faithfully encode biological theory. Risto Miikkulainen sees biology in another light, relating examples of evolutionary algorithms and their startling creativity. Shifting from the biological to the mechanical, Ben Recht explores notions of robustness through a novel synthesis of machine intelligence and control theory. In a similar vein, Daniela Rus outlines a brief history of robotics as a prelude to her work on self-driving cars and other autonomous agents. Gurjeet Singh subsequently brings the topology of machine learning to life. Ilya Sutskever recounts the mysteries of unsupervised learning and the promise of attention models. Oriol Vinyals then turns to deep learning vis-a-vis sequence to sequence models and imagines computers that generate their own algorithms. To conclude, Reza Zadeh reflects on the history and evolution of machine learning as a field and the role Apache Spark will play in its future.

It is important to note the scope of this report can only cover so much ground. With just ten interviews, it far from exhaustive: Indeed, for every such interview, dozens of other theoreticians and practitioners successfully advance the field through their efforts and dedication. This report, its brevity notwithstanding, offers a glimpse into this exciting field through the eyes of its leading minds.

CHAPTER 1

Anima Anandkumar: Learning in Higher Dimensions

Anima Anandkumar is on the faculty of the EECS Department at the University of California Irvine. Her research focuses on high-dimensional learning of probabilistic latent variable models and the design and analysis of tensor algorithms.

Key Takeaways

- Modern machine learning involves large amounts of data and a large number of variables, which makes it a high dimensional problem.
- Tensor methods are effective at learning such complex high dimensional problems, and have been applied in numerous domains, from social network analysis, document categorization, genomics, and towards understanding the neuronal behavior in the brain.
- As researchers continue to grapple with complex, highly-dimensional problems, they will need to rely on novel techniques in non-convex optimization, in the many cases where convex techniques fall short.

Let's start with your background.

I have been fascinated with mathematics since my childhood—its uncanny ability to explain the complex world we live in. During my college days, I realized the power of algorithmic thinking in computer science and engineering. Combining these, I went on to complete a Ph.D. at Cornell University, then a short postdoc at MIT before moving to the faculty at UC Irvine, where I've spent the past six years.

During my Ph.D., I worked on the problem of designing efficient algorithms for distributed learning. More specifically, when multiple devices or sensors are collecting data, can we design communication and routing schemes that perform "in-network" aggregation to reduce the amount of data transported, and yet, simultaneously, preserve the information required for certain tasks, such as detecting an anomaly? I investigated these questions from a statistical viewpoint, incorporating probabilistic graphical models, and designed algorithms that significantly reduce communication requirements. Ever since, I have been interested in a range of machine learning problems.

Modern machine learning naturally occurs in a world of higher dimensions, generating lots of multivariate data in the process, including a large amount of noise. Searching for useful information hidden in this noise is challenging; it is like the proverbial needle in a haystack.

The first step involves modeling the relationships between the hidden information and the observed data. Let me explain this with an example. In a recommender system, the hidden information represents users' unknown interests and the observed data consist of products they have purchased thus far. If a user recently bought a bike, she is interested in biking/outdoors, and is more likely to buy biking accessories in the near future. We can model her interest as a hidden variable and infer it from her buying pattern. To discover such relationships, however, we need to observe a whole lot of buying patterns from lots of users—making this problem a big data one.

My work currently focuses on the problem of efficiently training such hidden variable models on a large scale. In such an unsupervised approach, the algorithm automatically seeks out hidden factors that drive the observed data. Machine learning researchers, by

and large, agree this represents one of the key unsolved challenges in our field.

I take a novel approach to this challenge and demonstrate how tensor algebra can unravel these hidden, structured patterns without external supervision. Tensors are higher dimensional extensions of matrices. Just as matrices can represent pairwise correlations, tensors can represent higher order correlations (more on this later). My research reveals that operations on higher order tensors can be used to learn a wide range of probabilistic latent variable models efficiently.

What are the applications of your method?

We have shown applications in a number of settings. For example, consider the task of categorizing text documents automatically without knowing the topics *a priori*. In such a scenario, the topics themselves constitute hidden variables that must be gleaned from the observed text. A possible solution might be to learn the topics using word frequency, but this naive approach doesn't account for the same word appearing in multiple contexts.

What if, instead, we look at the co-occurrence of pairs of words, which is a more robust strategy than single word frequencies. But why stop at pairs? Why not examine the co-occurrences of triplets of words and so on into higher dimensions? What additional information might these higher order relationships reveal? Our work has demonstrated that uncovering hidden topics using the popular Latent Dirichlet Allocation (LDA) requires third-order relationships; pairwise relationships are insufficient.

The above intuition is broadly applicable. Take networks for example. You might try to discern hidden communities by observing the interaction of their members, examples of which include friendship connections in social networks, buying patterns in recommender systems or neuronal connections in the brain. My research reveals the need to investigate at least at the level of "friends of friends" or higher order relationships to uncover hidden communities. Although such functions have been used widely before, we were the first to show the precise information they contain and how to extract them in a computationally efficient manner.

We can extend the notion of hidden variable models even further. Instead of trying to discover one hidden layer, we look to construct a

hierarchy of hidden variables instead. This approach is better suited to a certain class of applications, including, for example, modeling the evolutionary tree of species or understanding the hierarchy of disease occurrence in humans. The goal in this case is to learn both the hierarchical structure of the latent variables, as well as the parameters that quantify the effect of the hidden variables on the given observed data.

The resulting structure reveals the hierarchical groupings of the observed variables at the leaves and the parameters quantify the "strength" of the group effect on the observations at the leaf nodes. We then simplify this to finding a hierarchical tensor decomposition, for which we have developed efficient algorithms.

So why are tensors themselves crucial in these applications?

First, I should note these tensor methods aren't just a matter of theoretical interest; they can provide enormous speedups in practice and even better accuracy, evidence of which we're seeing already. Kevin Chen from Rutgers University gave a compelling talk at the recent NIPS workshop on the superiority of these tensor methods in genomics: It offered better biological interpretation and yielded a 100x speedup when compared to the traditional expectation maximization (EM) method.

Tensor methods are so effective because they draw on highly optimized linear algebra libraries and can run on modern systems for large scale computation. In this vein, my student, Furong Huang, has deployed tensor methods on Spark, and it runs much faster than the variational inference algorithm, the default for training probabilistic models. All in all, tensor methods are now embarrassingly parallel and easy to run at large scale on multiple hardware platforms.

Is there something about tensor math that makes it so useful for these high dimensional problems?

Tensors model a much richer class of data, allowing us to grapple with multirelational data– both spatial and temporal. The different modes of the tensor, or the different directions in the tensor, represent different kinds of data.

At its core, the tensor describes a richer algebraic structure than the matrix and can thereby encode more information. For context, think of matrices as representing rows and columns – a two-

dimensional array, in other words. Tensors extend this idea to multi-dimensional arrays.

A matrix, for its part, is more than just columns and rows. You can sculpt it to your purposes though the math of linear operations, the study of which is called linear algebra. Tensors build on these malleable forms and their study, by extension, is termed multilinear algebra.

Given such useful mathematical structures, how can we squeeze them for information? Can we design and analyze algorithms for tensor operations? Such questions require a new breed of proof techniques built around non-convex optimization.

What do you mean by convex and non-convex optimization?

The last few decades have delivered impressive progress in convex optimization theory and technique. The problem, unfortunately, is that most optimization problems are not by their nature convex.

Let me expand on the issue of convexity by example. Let's say you're minimizing a parabolic function in one dimension: if you make a series of local improvements (at any starting point in the parabola) you are guaranteed to reach the best possible value. Thus, local improvements lead to global improvements. This property even holds for convex problems in higher dimensions. Computing local improvements is relatively easy using techniques such as gradient descent.

The real world, by contrast, is more complicated than any parabola. It contains a veritable zoo of shapes and forms. This translates to parabolas far messier than their ideal counterparts: Any optimization algorithm that makes local improvements will inevitably encounter ridges, valleys and flat surfaces; it is constantly at risk of getting stuck in a valley or some other roadblock—never reaching its global optimum.

As the number of variables increases, the complexity of these ridges and valleys explodes. In fact, there can be an exponential number of points where algorithms based on local steps, such as gradient descent, become stuck. Most problems, including the ones on which I am working, encounter this hardness barrier.

How does your work address the challenge of non-convex optimization?

The traditional approach to machine learning has been to first define learning objectives and then to use standard optimization frameworks to solve them. For instance, when learning probabilistic latent variable models, the standard objective is to maximize likelihood, and then to use the expectation maximization (EM) algorithm, which conducts a local search over the objective function. However, there is no guarantee that EM will arrive at a good solution. As it searches over the objective function, what may seem like a global optimum might merely be a spurious local one. This point touches on the broader difficulty with machine learning algorithm analysis, including backpropagation in neural networks: we cannot guarantee where the algorithm will end up or if it will arrive at a good solution.

To address such concerns, my approach looks for alternative, easy to optimize, objective functions for any given task. For instance, when learning latent variable models, instead of maximizing the likelihood function, I have focused on the objective of finding a good spectral decomposition of matrices and tensors, a more tractable problem given the existing toolset. That is to say, the spectral decomposition of the matrix is the standard singular-value decomposition (SVD), and we already possess efficient algorithms to compute the best such decomposition.

Since matrix problems can be solved efficiently despite being non-convex, and given matrices are special cases of tensors, we decided on a new research direction: Can we design similar algorithms to solve the decomposition of tensors? It turns out that tensors are much more difficult to analyze and can be NP-hard. Given that, we took a different route and sought to characterize the set of conditions under which such a decomposition can be solved optimally. Luckily, these conditions turn out to be fairly mild in the context of machine learning.

How do these tensor methods actually help solve machine learning problems? At first glance, tensors may appear irrelevant to such tasks. Making the connection to machine learning demands one additional idea, that of relationships (or moments). As I noted earlier, we can use tensors to represent higher order relationships among variables. And by looking at these relationships, we can learn the parameters of the latent variable models efficiently.

So you're able to bring a more elegant representation to modeling higher-dimensional data. Is this generally applicable in any form of machine learning?

I feel like we have only explored the tip of the iceberg. We can use tensor methods for training a wide class of latent variable models, such as modeling topics in documents, communities in networks, Gaussian mixtures, mixtures of ranking models and so on. These models, on their face, seem unrelated. Yet, they are unified by the ability to translate statistical properties, such as the conditional independence of variables, into algebraic constraints on tensors. In all these models, suitable moment tensors (usually the third or fourth order correlations) are decomposed to estimate the model parameters consistently. Moreover, we can prove that this requires only a small (precisely, a low-order polynomial) amount of samples and computation to work well.

So far, I discussed using tensors for unsupervised learning. We have also demonstrated that tensor methods provide guarantees for training neural networks, which sit in the supervised domain. We are currently tackling even harder questions such as reinforcement learning, where the learner interacts with and possibly changes the environment he/she is trying to understand. In general, I believe using higher order relationships and tensor algebraic techniques holds promise across a range of challenging learning problems.

What's next on the theoretical side of machine learning research?

These are exciting times to be a researcher in machine learning. There is a whole spectrum of problems ranging from fundamental research to real-world at scale deployment. I have been pursuing research from an interdisciplinary lens; by combining tensor algebra with probabilistic modeling, we have developed a completely new set of learning algorithms with strong theoretical guarantees. I believe making such non-obvious connections is crucial towards breaking the hardness barriers in machine learning.

CHAPTER 2

Yoshua Bengio: Machines That Dream

Yoshua Bengio is a professor with the department of computer science and operations research at the University of Montreal, where he is head of the Machine Learning Laboratory (MILA) and serves as the Canada Research Chair in statistical learning algorithms. The goal of his research is to understand the principles of learning that yield intelligence.

Key Takeaways

- Natural language processing has come a long way since its inception. Through techniques such as vector representation and custom deep neural nets, the field has taken meaningful steps towards real language understanding.
- The language model endorsed by deep learning breaks with the Chomskyan school and harkens back to Connectionism, a field made popular in the 1980s.
- In the relationship between neuroscience and machine learning, inspiration flows both ways, as advances in each respective field shine new light on the other.
- Unsupervised learning remains one of the key mysteries to be unraveled in the search for true AI. A measure of our progress

towards this goal can be found in the unlikeliest of places—inside the machine's dreams.

Let's start with your background.

I have been researching neural networks since the 80s. I got my Ph.D. in 1991 from McGill University, followed by a postdoc at MIT with Michael Jordan. Afterward, I worked with Yann LeCun, Patrice Simard, Léon Bottou, Vladimir Vapnik, and others at Bell Labs and returned to Montreal, where I've spent most my life.

As fate would have it, neural networks fell out of fashion in the mid-90s, re-emerging only in the last decade. Yet throughout that period, my lab, alongside a few other groups pushed forward. And then, in a breakthrough around 2005 or 2006, we demonstrated the first way to successfully train deep neural nets, which had resisted previous attempts.

Since then, my lab has grown into its own institute with five or six professors and totaling about 65 researchers. In addition to advancing the area of unsupervised learning, over the years, our group has contributed to a number of domains, including, for example, natural language, as well as recurrent networks, which are neural networks designed specifically to deal with sequences in language and other domains.

At the same time, I'm keenly interested in the bridge between neuroscience and deep learning. Such a relationship cuts both ways. On the one hand, certain currents in AI research dating back to the very beginning of AI in the 50s, draw inspiration from the human mind. Yet ever since neural networks have re-emerged in force, we can flip this idea on its head and look to machine learning instead as an inspiration to search for high-level theoretical explanations for learning in the brain.

Let's move on to natural language. How has the field evolved?

I published my first big paper on natural language processing in 2000 at the NIPS Conference. Common wisdom suggested the state-of-the-art language processing approaches of this time would never deliver AI because it was, to put it bluntly, too dumb. The basic technique in vogue at the time was to count how many times, say, a word is followed by another word, or a sequence of three words come

together—so as to predict the next word or translate a word or phrase.

Such an approach, however, lacks any notion of meaning, precluding its application to highly complex concepts and generalizing correctly to sequences of words that had not been previously seen. With this in mind, I approached the problem using neural nets, believing they could overcome the "curse of dimensionality" and proposed a set of approaches and arguments that have since been at the heart of deep learning's theoretical analysis.

This so-called curse speaks to one of fundamental challenges in machine learning. When trying to predict something using an abundance of variables, the huge number of possible combinations of values they can take makes the problem exponentially hard. For example, if you consider a sequence of three words and each word is one out of a vocabulary of 100,000, how many possible sequences are there? 100,000 to the cube, which is much more than the number of such sequences a human could ever possibly read. Even worse, if you consider sequences of 10 words, which is the scope of a typical short sentence, you're looking at 100,000 to the power of 10, an unthinkably large number.

Thankfully, we can replace words with their representations, otherwise known as word vectors, and learn these word vectors. Each word maps to a vector, which itself is a set of numbers corresponding to automatically learned attributes of the word; the learning system simultaneously learns using these attributes of each word, for example to predict the next word given the previous ones or to produce a translated sentence. Think of the set of word vectors as a big table (number of words by number of attributes) where each word vector is given by a few hundred attributes. The machine ingests these attributes and feeds them as an input to a neural net. Such a neural net looks like any other traditional net except for its many outputs, one per word in the vocabulary. To properly predict the next word in a sentence or determine the correct translation, such networks might be equipped with, say, 100,000 outputs.

This approach turned out to work really well. While we started testing this at a rather small scale, over the following decade, researchers have made great progress towards training larger and larger models on progressively larger datasets. Already, this technique is displacing a number of well-worn NLP approaches, consistently

besting state-of-the-art benchmarks. More broadly, I believe we're in the midst of a big shift in natural language processing, especially as it regards semantics. Put another way, we're moving towards natural language *understanding*, especially with recent extensions of recurrent networks that include a form of *reasoning*.

Beyond its immediate impact in NLP, this work touches on other, adjacent topics in AI, including how machines answer questions and engage in dialog. As it happens, just a few weeks ago, DeepMind published a paper in *Nature* on a topic closely related to deep learning for dialogue. Their paper describes a deep reinforcement learning system that beat the European Go champion. By all accounts, Go is a very difficult game, leading some to predict it would take decades before computers could face off against professional players. Viewed in a different light, a game like Go looks a lot like a conversation between the human player and the machine. I'm excited to see where these investigations lead.

How does deep learning accord with Noam Chomsky's view of language?

It suggests the complete opposite. Deep learning relies almost completely on learning through data. We of course design the neural net's architecture, but for the most part, it relies on data and lots of it. And whereas Chomsky focused on an innate grammar and the use of logic, deep learning looks to meaning. Grammar, it turns out, is the icing on the cake. Instead, what really matters is our intention: it's mostly the choice of words that determines what we mean, and the associated meaning can be learned. These ideas run counter to the Chomskyan school.

Is there an alternative school of linguistic thought that offers a better fit?

In the '80s, a number of psychologists, computer scientists and linguists developed the Connectionist approach to cognitive psychology. Using neural nets, this community cast a new light on human thought and learning, anchored in basic ingredients from neuroscience. Indeed, backpropagation and some of the other algorithms in use today trace back to those efforts.

Does this imply that early childhood language development or other functions of the human mind might be structurally similar to backprop or other such algorithms?

Researchers in our community sometimes take cues from nature and human intelligence. As an example, take curriculum learning. This approach turns out to facilitate deep learning, especially for reasoning tasks. In contrast, traditional machine learning stuffs all the examples in one big bag, making the machine examine examples in a random order. Humans don't learn this way. Often with the guidance of a teacher, we start with learning easier concepts and gradually tackle increasingly difficult and complex notions, all the while building on our previous progress.

From an optimization point of view, training a neural net is difficult. Nevertheless, by starting small and progressively building on layers of difficulty, we can solve the difficult tasks previously considered too difficult to learn.

Your work includes research around deep learning architectures. Can you touch on how those have evolved over time?

We don't necessarily employ the same kind of nonlinearities as we used in the '80s through the first decade of 2000. In the past, we relied on, for example, the hyperbolic tangent, which is a smoothly increasing curve that saturates for both small and large values, but responds to intermediate values. In our work, we discovered that another nonlinearity, hiding in plain sight, the rectifier, allowed us to train much deeper networks. This model draws inspiration from the human brain, which fits the rectifier more closely than the hyperbolic tangent. Interestingly, the reason it works as well as it does remains to be clarified. Theory often follows experiment in machine learning.

What are some of the other challenges you hope to address in the coming years?

In addition to understanding natural language, we're setting our sights on reasoning itself. Manipulating symbols, data structures and graphs used to be realm of classical AI (sans learning), but in just the past few years, neural nets re-directed to this endeavor. We've seen models that can manipulate data structures like stacks and graphs, use memory to store and retrieve objects and work through a sequence of steps, potentially supporting dialog and other tasks that depend on synthesizing disparate evidence.

In addition to reasoning, I'm very interested in the study of unsupervised learning. Progress in machine learning has been driven, to

a large degree, by the benefit of training on massive data sets with millions of labeled examples, whose interpretation has been tagged by humans. Such an approach doesn't scale: We can't realistically label everything in the world and meticulously explain every last detail to the computer. Moreover, it's simply not how humans learn most of what they learn.

Of course, as thinking beings, we offer and rely on feedback from our environment and other humans, but it's sparse when compared to your typical labeled dataset. In abstract terms, a child in the world observes her environment in the process of seeking to understand it and the underlying causes of things. In her pursuit of knowledge, she experiments and asks questions to continually refine her internal model of her surroundings.

For machines to learn in a similar fashion, we need to make more progress in unsupervised learning. Right now, one of the most exciting areas in this pursuit centers on generating images. One way to determine a machine's capacity for unsupervised learning is to present it with many images, say, of cars, and then to ask it to "dream" up a novel car model—an approach that's been shown to work with cars, faces, and other kinds of images. However, the visual quality of such dream images is rather poor, compared to what computer graphics can achieve.

If such a machine responds with a reasonable, non-facsimile output to such a request to generate a new but plausible image, it suggests an understanding of those objects a level deeper: In a sense, this machine has developed an understanding of the underlying explanations for such objects.

You said you ask the machine to dream. At some point, it may actually be a legitimate question to ask…do androids dream of electric sheep, to quote Philip K. Dick?

Right. Our machines already dream, but in a blurry way. They're not yet crisp and content-rich like human dreams and imagination, a facility we use in daily life to imagine those things which we haven't actually lived. I am able to imagine the consequence of taking the wrong turn into oncoming traffic. I thankfully don't need to actually live through that experience to recognize its danger. If we, as humans, could solely learn through supervised methods, we would need to explicitly experience that scenario and endless permutations thereof. Our goal with research into unsupervised learning is to help

the machine, given its current knowledge of the world reason and predict what will probably happen in its future. This represents a critical skill for AI.

It's also what motivates science as we know it. That is, the methodical approach to discerning causal explanations for given observations. In other words, we're aiming for machines that function like little scientists, or little children. It might take decades to achieve this sort of true autonomous unsupervised learning, but it's our current trajectory.

CHAPTER 3

Brendan Frey: Deep Learning Meets Genome Biology

Brendan Frey is a co-founder of Deep Genomics, a professor at the University of Toronto and a co-founder of its Machine Learning Group, a senior fellow of the Neural Computation program at the Canadian Institute for Advanced Research and a fellow of the Royal Society of Canada. His work focuses on using machine learning to understand the genome and to realize new possibilities in genomic medicine.

Key Takeaways

- The application of deep learning to genomic medicine is off to a promising start; it could impact diagnostics, intensive care, pharmaceuticals and insurance.
- The "genotype-phenotype divide"—our inability to connect genetics to disease phenotypes—is preventing genomics from advancing medicine to its potential.
- Deep learning can bridge the genotype-phenotype divide, by incorporating an exponentially growing amount of data, and accounting for the multiple layers of complex biological processes that relate the genotype to the phenotype.
- Deep learning has been successful in applications where humans are naturally adept, such as image, text, and speech understanding. The human mind, however, isn't intrinsically

designed to understand the genome. This gap necessitates the application of "super-human intelligence" to the problem.

- Efforts in this space must account for underlying biological mechanisms; overly simplistic, "black box" approaches will drive only limited value.

Let's start with your background.

I completed my Ph.D. with Geoff Hinton in 1997. We co-authored one of the first papers on deep learning, published in *Science* in 1995. This paper was a precursor to much of the recent work on unsupervised learning and autoencoders. Back then, I focused on computational vision, speech recognition and text analysis. I also worked on message passing algorithms in deep architectures. In 1997, David MacKay and I wrote one of the first papers on "loopy belief propagation" or the "sum-product algorithm," which appeared in the top machine learning conference, the Neural Information Processing Systems Conference, or NIPS.

In 1999, I became a professor of Computer Science at the University of Waterloo. Then in 2001, I joined the University of Toronto and, along with several other professors, co-founded the Machine Learning Group. My team studied learning and inference in deep architectures, using algorithms based on variational methods, message passing and Markov chain Monte Carlo (MCMC) simulation. Over the years, I've taught a dozen courses on machine learning and Bayesian networks to over a thousand students in all.

In 2005, I became a senior fellow in the Neural Computation program of the Canadian Institute for Advanced Research, an amazing opportunity to share ideas and collaborate with leaders in the field, such as Yann LeCun, Yoshua Bengio, Yair Weiss, and the Director, Geoff Hinton.

What got you started in genomics?

It's a personal story. In 2002, a couple years into my new role as a professor at the University of Toronto, my wife at the time and I learned that the baby she was carrying had a genetic problem. The counselor we met didn't do much to clarify things: she could only suggest that either nothing was wrong, or that, on the other hand, something may be terribly wrong. That experience, incredibly difficult for many reasons, also put my professional life into sharp relief:

the mainstay of my work, say, in detecting cats in YouTube videos, seemed less significant—all things considered.

I learned two lessons: first, I wanted to use machine learning to improve the lives of hundreds of millions of people facing similar genetic challenges. Second, reducing uncertainty is tremendously valuable: Giving someone news, either good or bad, lets them plan accordingly. In contrast, uncertainty is usually very difficult to process.

With that, my research goals changed in kind. Our focus pivoted to understanding how the genome works using deep learning.

Why do you think machine learning plus genome biology is important?

Genome biology, as a field, is generating torrents of data. You will soon be able to sequence your genome using a cell-phone size device for less than a trip to the corner store. And yet the genome is only part of the story: there exists huge amounts of data that describe cells and tissues. We, as humans, can't quite grasp all this data: We don't yet know enough biology. Machine learning can help solve the problem.

At the same time, others in the machine learning community recognize this need. At last year's premier conference on machine learning, four panelists, Yann LeCun, Director of AI at Facebook, Demis Hassabis, co-founder of DeepMind, Neil Lawrence, Professor at the University of Sheffield, and Kevin Murphy from Google, identified medicine as the next frontier for deep learning.

To succeed, we need to bridge the "genotype-phenotype divide." Genomic and phenotype data abound. Unfortunately, the state-of-the-art in meaningfully connecting these data results in a slow, expensive and inaccurate process of literature searches and detailed wetlab experiments. To close the loop, we need systems that can determine intermediate phenotypes called "molecular phenotypes," which function as stepping stones from genotype to disease phenotype. For this, machine learning is indispensable.

As we speak, there's a new generation of young researchers using machine learning to study how genetics impact molecular phenotypes, in groups such as Anshul Kundaje's at Stanford. To name just a few of these upcoming leaders: Andrew Delong, Babak Alipanahi and David Kelley of the University of Toronto and Harvard, who study protein-DNA interactions; Jinkuk Kim of MIT who studies

gene repression; and Alex Rosenberg, who is developing experimental methods for examining millions of mutations and their influence on splicing at the University of Washington. In parallel, I think it's exciting to see an emergence of startups working in this field, such as Atomwise, Grail and others.

What was the state of the genomics field when you started to explore it?

Researchers used a variety of simple "linear" machine learning approaches, such as support vector machines and linear regression that could, for instance, predict cancer from a patient's gene expression pattern. These techniques were by their design, "shallow." In other words, each input to the model would net a very simple "advocate" or "don't advocate" for the class label. Those methods didn't account for the complexity of biology.

Hidden Markov models and related techniques for analyzing sequences became popular in the 1990's and early 2000's. Richard Durbin and David Haussler were leading groups in this area. Around the same time, Chris Burge's group at MIT developed a Markov model that could detect genes, inferring the beginning of the gene as well as the boundaries between different parts, called introns and exons. These methods were useful for low-level "sequence analysis", but they did not bridge the genotype-phenotype divide.

Broadly speaking, the state of research at the time was driven by primarily shallow techniques that did not sufficiently account for the underlying biological mechanisms for how the text of the genome gets converted into cells, tissues and organs.

What does it mean to develop computational models that sufficiently account for the underlying biology?

One of the most popular ways of relating genotype to phenotype is to look for mutations that correlate with disease, in what's called a genome-wide association study (GWAS). This approach is also shallow in the sense that it discounts the many biological steps involved in going from a mutation to the disease phenotype. GWAS methods can identify regions of DNA that may be important, but most of the mutations they identify aren't causal. In most cases, if you could "correct" the mutation, it wouldn't affect the phenotype.

A very different approach accounts for the intermediate molecular phenotypes. Take gene expression, for example. In a living cell, a

gene gets expressed when proteins interact in a certain way with the DNA sequence upstream of the gene, i.e., the "promoter." A computational model that respects biology should incorporate this promoter-to-gene expression chain of causality. In 2004, Beer and Tavazoie wrote what I considered an inspirational paper. They sought to predict every yeast gene's expression level based on its promoter sequence, using logic circuits that took as input features derived from the promoter sequence. Ultimately, their approach didn't pan out, but was a fascinating endeavor nonetheless.

My group's approach was inspired by Beer and Tavazoie's work, but differed in three ways: we examined mammalian cells; we used more advanced machine learning techniques; and we focused on splicing instead of transcription. This last difference was a fortuitous turn in retrospect. Transcription is far more difficult to model than splicing. Splicing is a biological process wherein some parts of the gene (introns) are removed and the remaining parts (exons) are connected together. Sometimes exons are removed too, and this can have a major impact on phenotypes, including neurological disorders and cancers.

To crack splicing regulation using machine learning, my team collaborated with a group led by an excellent experimental biologist named Benjamin Blencowe. We built a framework for extracting biological features from genomic sequences, pre-processing the noisy experimental data, and training machine learning techniques to predict splicing patterns from DNA. This work was quite successful, and led to several publications in *Nature* and *Science*.

Is genomics different from other applications of machine learning?

We discovered that genomics entails unique challenges, compared to vision, speech and text processing. A lot of the success in vision rests on the assumption that the object to be classified occupies a substantial part of the input image. In genomics, the difficulty emerges because the object of interest occupies only a tiny fraction—say, one millionth—of the input. Put another way, your classifier acts on trace amounts of signal. Everything else is noise—and lots of it. Worse yet, it's relatively structured noise comprised of other, much larger objects irrelevant to the classification task. That's genomics for you.

The more concerning complication is that we don't ourselves really know how to interpret the genome. When we inspect a typical

image, we naturally recognize its objects and by extension, we know what we want the algorithm to look for. This applies equally well to text analysis and speech processing, domains in which we have some handle on the truth. In stark contrast, humans are not naturally good at interpreting the genome. In fact, they're very bad at it. All this is to say that we must turn to truly superhuman artificial intelligence to overcome our limitations.

Can you tell us more about your work around medicine?

We set out to train our systems to predict molecular phenotypes without including any disease data. Yet once it was trained, we realized our system could in fact make accurate predictions for disease; it learned how the cell reads the DNA sequence and turns it into crucial molecules. Once you have a computational model of how things work normally, you can use it to detect when things go awry.

We then directed our system to large scale disease mutation datasets. Suppose there is some particular mutation in the DNA. We feed that mutated DNA sequence, as well as its non-mutated counterpart, into our system and compare the two outputs, the molecular phenotypes. If we observe a big change, we label the mutation as potentially pathogenic. It turns out that this approach works well.

But of course, it isn't perfect. First, the mutation may change the molecular phenotype, but not lead to disease. Second, the mutation may not affect the molecular phenotype that we're modeling, but lead to a disease in some other way. Third, of course, our system isn't perfectly accurate. Despite these shortcomings, our approach can accurately differentiate disease from benign mutations. Last year, we published papers in *Science* and *Nature Biotechnology* demonstrating that the approach is significantly more accurate than competing ones.

Where is your company, Deep Genomics, headed?

Our work requires specialized skills from a variety of areas, including deep learning, convolutional neural networks, random forests, GPU computing, genomics, transcriptomics, high-throughput experimental biology, and molecular diagnostics. For instance, we have on board Hui Xiong, who invented a Bayesian deep learning algorithm for predicting splicing, and Daniele Merico, who developed the whole genome sequencing diagnostics system used at the

Hospital for Sick Children. We will continue to recruit talented people in these domains.

Broadly speaking, our technology can impact medicine in numerous ways, including: Genetic diagnostics, refining drug targets, pharmaceutical development, personalized medicine, better health insurance and even synthetic biology. Right now, we are focused on diagnostics, as it's a straightforward application of our technology. Our engine provides a rich source of information that can be used to make more reliable patient decisions at lower cost.

Going forward, many emerging technologies in this space will require the ability to understand the inner workings of the genome. Take, for example, gene editing using the CRISPR/Cas9 system. This technique let's us "write" to DNA and as such could be a very big deal down the line. That said, knowing *how* to write is not the same as knowing *what* to write. If you edit DNA, it may make the disease worse, not better. Imagine instead if you could use a computational "engine" to determine the consequences of gene editing writ large? That is, to be fair, a ways off. Yet ultimately, that's what we want to build.

CHAPTER 4

Risto Miikkulainen: Stepping Stones and Unexpected Solutions in Evolutionary Computing

Risto Miikkulainen is professor of computer science and neuroscience at the University of Texas at Austin, and a fellow at Sentient Technologies, Inc. Risto's work focuses on biologically inspired computation such as neural networks and genetic algorithms.

Key Takeaways

- Evolutionary computation is a form of reinforcement learning applied to optimizing a fitness function.
- Its applications include robotics, software agents, design, and web commerce.
- It enables the discovery of truly novel solutions.

Let's start with your background.

I completed my Ph.D. in 1990 at the UCLA computer science department. Following that, I became a professor in the computer science department at the University of Texas, Austin. My dissertation and early work focused on building neural network models of cognitive science—language processing and memory, in particular. That work has continued throughout my career. I recently dusted off

those models to drive towards understanding cognitive dysfunction like schizophrenia and aphasia in bilinguals.

Neural networks, as they relate to cognitive science and engineering, have been a main focus throughout my career. In addition to cognitive science, I spent a lot of time working in computational neuroscience.

More recently, my team and I have been focused on neuroevolution; that is, optimizing neural networks using evolutionary computation. We have discovered that neuroevolution research involves a lot of the same challenges as cognitive science, for example, memory, learning, communication and so on. Indeed, these fields are really starting to come together.

Can you give some background on how evolutionary computation works, and how it intersects with deep learning?

Deep learning is a supervised learning method on neural networks. Most of the work involves supervised applications where you already know what you want, e.g., weather predictions, stock market prediction, the consequence of a certain action when driving a car. You are, in these cases, learning a nonlinear statistical model of that data, which you can then re-use in future situations. The flipside of that approach concerns unsupervised learning, where you learn the structure of the data, what kind of clusters there are, what things are similar to other things. These efforts can provide a useful internal representation for a neural network.

A third approach is called reinforcement learning. Suppose you are driving a car or playing a game: It's harder to define the optimal actions, and you don't receive much feedback. In other words, you can play the whole game of chess, and by the end, you've either won or lost. You know that if you lost, you probably made some poor choices. But which? Or, if you won, which were the well-chosen actions? This is, in a nutshell, a reinforcement learning problem.

Put another way, in this paradigm, you receive feedback periodically. This feedback, furthermore, will only inform you about how well you did without in turn listing the optimal set of steps or actions you took. Instead, you have to discover those actions through exploration—testing diverse approaches and measuring their performance.

Enter evolutionary computation, which can be posed as a way of solving reinforcement learning problems. That is, there exists some fitness function, and you focus on evolving a solution that optimizes that function.

In many cases, however, in the real world, you do not have a full state description—a full accounting of the facts on the ground at any given moment. You don't, in other words, know the full context of your surroundings. To illustrate this problem, suppose you are in a maze. Many corridors look the same to you. If you are trying to learn to associate a value for each action/state pair, and you don't know what state you are in, you cannot learn. This is the main challenge for reinforcement learning approaches that learn such utility values for each action in each respective state.

Evolutionary computation, on the other hand, can be very effective in addressing these problems. In this approach, we use evolution to construct a neural network, which then ingests the state representation, however noisy or incomplete, and suggests an action that is most likely to be beneficial, correct, or effective. It doesn't need to learn values for each action in each state. It always has a complete policy of what to do—evolution simply refines that policy. For instance, it might first, say, always turn left at corners and avoid walls, and gradually then evolve towards other actions as well. Furthermore, the network can be recurrent, and consequently remember how it "got" to that corridor, which disambiguates the state from other states that look the same. Neuroevolution can perform better on problems where part of the state is hidden, as is the case in many real-world problems.

How formally does evolutionary computation borrow from biology, and how you are driving toward potentially deepening that metaphor?

Some machine learning comprises pure statistics or is otherwise mathematics-based, but some of the inspiration in evolutionary computation, and in neural networks and reinforcement learning in general, does in fact derive from biology. To your question, it is indeed best understood as a metaphor; we aren't systematically replicating what we observe in the biological domain. That is, while some of these algorithms are inspired by genetic evolution, they don't yet incorporate the overwhelming complexity of genetic expression, epigenetic influence and the nuanced interplay of an organism with its environment.

Instead, we take the aspects of biological processes that make computational sense and translate them into a program. The driving design of this work, and indeed the governing principle of biological evolution, can be understood as selection on variation.

At a high level, it's quite similar to the biological story. We begin with a population from which we select the members that reproduce the most, and through selective pressure, yield a new population that is more likely to be better than the previous one. In the meantime, researchers are working on incorporating increasing degrees of biological complexity into these models. Much work remains to be done in this regard.

What are some applications of this work?

Evolutionary algorithms have existed for quite a while, indeed since the '70s. The lion's share of work centered around engineering applications, e.g., trying to build better power grids, antennas and robotic controllers through various optimization methods. What got us really excited about this field are the numerous instances where evolution not only optimizes something that you know well, but goes one step further and generates novel and indeed surprising solutions.

We encountered such a breakthrough when evolving a controller for a robot arm. The arm had six degrees of freedom, although you really only needed three to control it. The goal was to get its fingers to a particular location in 3D space. This was a rather straightforward exercise, so we complicated things by inserting obstacles along its path, all the while evolving a controller that would get to the goal while avoiding said obstacles. One day while working on this problem, we accidentally disabled the main motor, i.e., the one that turns the robot around its main axis. Without that particular motor, it could not reach its goal location.

We ran the evolution program, and although it took five times longer than usual, it ultimately found a solution that would guide the fingers into the intended location. We only understood what was going on when we looked at a graphical visualization of its behavior. When the target was, say, all the way to the left, and the robot needed to turn around the main axis to get its arm into close proximity – it was, by definition, unable to turn without its main motor. Instead, it turned the arm from the elbow and the shoulder, *away* from the goal, then swung it back with quite some force. Thanks to

momentum, the robot would turn around its main axis, and get to the goal location, even without the motor. This was surprising to say the least.

This is exactly what you want in a machine learning system. It fundamentally innovates. If a robot on Mars loses its wheel or gets stuck on a rock, you still want it to creatively complete its mission.

Let me further underscore this sort of emergent creativity with another example (of which there are many!). in one of my classes, we assigned students to build a game-playing agent to win a game similar to tic-tac-toe, only played on a very large grid where the goal is to get five in a row. The class developed a variety of approaches, including neural networks and some rule-based systems, but the winner was an evolution system that evolved to make the first move to a location really far away, millions of spaces away from where the game play began. Opposing players would then expand memory to capture that move, until they ran out of memory and crashed. It was a very creative way of winning, something that you might not have considered *a priori*.

Evolution thrives on diversity. If you supply it with representations and allow it to explore a wide space, it can discover solutions that are truly novel and interesting. In deep learning, most of the time you are learning a task you already know—weather prediction, stock market prediction, etc.—but, here, we are being creative. We are not just predicting what will happen, but we are creating objects that didn't previously exist.

What is the practical application of this kind of learning in industry? You mentioned the Mars rover, for example, responding to some obstacle with evolution-driven ingenuity. Do you see robots and other physical or software agents being programmed with this sort of on the fly, ad hoc, exploratory creativity?

Sure. We have shown that evolution works. We're now focused on taking it out into the world and matching it to relevant applications. Robots, for example, are a good use case: They have to be safe; they have to be robust; and they have to work under conditions that no-one can fully anticipate or model. An entire branch of AI called evolutionary robotics centers around evolving behaviors for these kinds of real, physical robots.

At the same time, evolutionary approaches can be useful for software agents, from virtual reality to games and education. Many systems and use cases can benefit from the optimization and creativity of evolution, including web design, information security, optimizing traffic flow on freeways or surface roads, optimizing the design of buildings, computer systems, and various mechanical devices, as well as processes such as bioreactors and 3-D printing. We're beginning to see these applications emerge.

What would you say is the most exciting direction of this research?

I think it is the idea that, in order to build really complex systems, we need to be able to use "stepping stones" in evolutionary search. It is still an open question: using novelty, diversity and multiple objectives, how do we best discover components that can be used to construct complex solutions? That is crucial in solving practical engineering problems such as making a robot run fast or making a rocket fly with stability, but also in constructing intelligent agents that can learn during their lifetime, utilize memory effectively, and communicate with other agents.

But equally exciting is the emerging opportunity to take these techniques to the real world. We now have plenty of computational power, and evolutionary algorithms are uniquely poised to take advantage of it. They run in parallel and can as a result operate at very large scale. The upshot of all of this work is that these approaches can be successful on large-scale problems that cannot currently be solved in any other way.

CHAPTER 5

Benjamin Recht: Machine Learning in the Wild

Benjamin Recht is an associate professor in the electrical engineering and computer sciences department and the statistics department at the University of California at Berkeley. His research focuses on scalable computational tools for large-scale data analysis, statistical signal processing, and machine learning—exploring the intersections of convex optimization, mathematical statistics, and randomized algorithms.

Key Takeaways

- Machine learning can be effectively related to control theory, a field with roots in the 1950s.
- In general, machine learning looks to make predictions by training on vast amounts of data to predict the average case. On the other hand, control theory looks to build a physical model of reality and warns of the worst case (i.e., this is how the plane responds to turbulence).
- Combining control principles with reinforcement learning will enable machine learning applications in areas where the worst case can be a question of life or death (e.g., self driving cars).

You're known for thinking about computational issues in machine learning, but you've recently begun to relate it to control theory. Can you talk about some of that work?

I've written a paper with Andy Packard and Laurent Lessard, two control theorists. Control theory is most commonly associated with aviation or manufacturing. So you might think, what exactly does autopilot have to do with machine learning? We're making great progress in machine learning systems, and we're trying to push their tenets into many different kinds of production systems. But we're doing so with limited knowledge about how well these things are going to perform in the wild.

This isn't such a big deal with most machine learning algorithms that are currently very successful. If image search returns an outlier, it's often funny or cute. But when you put a machine learning system in a self-driving car, one bad decision can lead to serious human injury. Such risks raise the stakes for the safe deployment of learning systems.

Can you explain how terms like *robustness* and *error* are defined in control system theory?

In engineering design problems, robustness and performance are competing objectives. Robustness means having repeatable behavior no matter what the environment is doing. On the other hand, you want this behavior to be as good as possible. There are always some performance goals you want the system to achieve. Performance is a little bit easier to understand—faster, more scalable, higher accuracy, etc. Performance and robustness trade off with each other: the most robust system is the one that does nothing, but the highest performing systems typically require sacrificing some degree of safety.

Can you share some examples and some of the theoretical underpinnings of the work and your recent paper?

The paper with Laurent and Andy noted that all of the algorithms we popularly deploy in machine learning look like classic dynamical systems that control theorists have studied since the 1950's. Once we drew the connection, we realized we could lean on 70 years of experience analyzing these systems. Now we can examine how these machine learning algorithms perform as you add different kinds of noise and interference to their execution.

For one very popular algorithm called the Heavy Ball method, we discovered that if you use off-the-shelf settings, there are cases when it never converges. No one had yet produced a formal proof that the algorithm converged, but everybody assumed it worked in practice. Moreover, we were able to modify the parameters to find a regime where it always converged. What makes this analysis toolkit so useful is that we can not only certify whether a method will work, but we can interactively manipulate a specified algorithm to make it more robust.

Do you mean I can take a library of linear and nonlinear algorithms, supervised and unsupervised approaches, and basically score them according to how robust they are?

Yes. We've only done this in some very simple cases so far, but we're hoping to expand on this work. You can plug the algorithm into this framework, and we'll give you back an analysis as to how fast it might converge or how much noise it can reject. Then you can tune this algorithm to improve some metric of interest.

Control systems that might, for example, model airplane flight, don't derive their parameters by studying millions of hours of flight in the way we understand a classical machine learning algorithm might. How do control theorists build their models in contrast to machine learning approaches?

Control is very much about building reasonable models based on understanding how a system responds to different conditions. Air passes over a wing, which will create some kind of lift. They work from these physics models of aerodynamics and then they build a control system around that to make sure you actually fly in a straight line. Now, things get complicated when you add in turbulence, but rather than build a more complicated model of turbulence here, they model this as a "black-box" disturbance. Control theory aims to build policies that keep the plane up in the air as long as the black-box disturbance isn't too extreme.

In machine learning, I would like to decide whether or not there's a human in front of me if, for example, I'm a self-driving car. I might use a dictionary of 15 million images, some of them labeled with "human" and some of them labeled with "not human." My model is derived from this huge data set rather than from physical principles about how humans present themselves in a scene. One of the guiding principles of machine learning is that if you give me all the data

in the universe, then I can make any prediction you need. This is also one of its main conceits.

Right. Turbulence is not predictable, but it *is* kind of predictable. It's predictable insofar as how the plane is going to respond. So control systems are, in a way, more deterministic.

Yes, exactly. Turbulence is exactly the idea of robustness. So, you can either apply a model to turbulence, or you can just look for the worst case outcome that can happen under turbulent behavior. The latter is much easier. That's what robust control people do. You take your uncertainty, you try to put it in a box, and you say, "That's what uncertainty looks like."

Now, you can build control systems without physical models. Look at what the guys at DeepMind are doing with video games. They are using techniques from reinforcement learning to outperform humans. In reinforcement learning, rather than building a model, you just play a lot of representative scenes to a control system, and you modify the controller after each interaction in such a way that you improve the performance. That's how the machines learn to play Atari games. They just play it thousands and thousands and thousands of times and make a record of every possible thing you could do in this Atari game and then build a data-driven control policy from there. My colleague Pieter Abbeel and his students have recently made some remarkable progress using reinforcement learning and neural networks to learn locomotion and to enable robots to nimbly interact with real objects.

Is there a difference between how control theorists and machine learning researchers think about robustness and error?

In machine learning, we almost always model our errors as being random rather than worst-case. In some sense, random errors are actually much more benign than worst-case errors. Let's say you're just going to add up a sequence of numbers. Each number is either one or minus one, and we're going to sum up 20 of them. The worst case sum—that is the largest sum—is achieved when you set all of your choices equal to one. This gets you 20. But if you flip a coin to assign the ones and minus ones, on average the sum will be zero! And, more often than not, you'll get something on the order of five. It will be consistently smaller. The odds of getting a 20 is one in a million.

In machine learning, by assuming average-case performance, rather than worst-case, we can design predictive algorithms by averaging out the errors over large data sets. We want to be robust to fluctuations in the data, but only *on average*. This is much less restrictive than the worst-case restrictions in controls

This ties back to your earlier point about average versus worst/best case.

Exactly. It's a huge deal. We don't want to rely solely on worst-case analysis because that's not going to reflect our reality. On the other hand, it would be good to have at least more robustness in our predictions and a little bit of understanding about how these are going to fare as our data varies and our data changes.

One example where my collaborators and I have been able to take advantage of randomness came in a study of stochastic gradient descent (SGD). SGD is probably the most popular algorithm in machine learning, and is the foundation of how we train neural nets. Feng Niu, Chris Re, Stephen Wright, and I were able to parallelize this algorithm by taking advantage of randomness. Feng, a grad student at the time, was experimenting with some strategies to parallelize SGD. Out of frustration, he turned off the locks in his parallel code. To all of our surprise, it just worked better. It worked a lot better. Basically, we started getting linear speedups.

In trying to explain that phenomenon, we formalized something called "HOGWILD!"—a lock-free approach to parallelizing stochastic gradient descent. In the worst case, the HOGWILD! approach would degrade performance. But because the errors are random, you get dramatic speedups in practice. People picked up on the idea and started implementing it. And for a lot of the state-of-the-art deep learning models, HOGWILD! became a go-to technique.

So, control theory is model-based and concerned with worst case. Machine learning is data based and concerned with average case. Is there a middle ground?

I think there is! And I think there's an exciting opportunity here to understand how to combine robust control and reinforcement learning. Being able to build systems from data alone simplifies the engineering process, and has had several recent promising results. Guaranteeing that these systems won't behave catastrophically will enable us to actually deploy machine learning systems in a variety of applications with major impacts on our lives. It might enable safe

autonomous vehicles that can navigate complex terrains. Or could assist us in diagnostics and treatments in health care. There are a lot of exciting possibilities, and that's why I'm excited about how to find a bridge between these two viewpoints.

CHAPTER 6

Daniela Rus: The Autonomous Car As a Driving Partner

Daniela Rus is a professor of electrical engineering and computer science and director of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. Rus is the first woman to serve as director of CSAIL and its predecessors—the AI Lab and the Lab for Computer Science.

Key Takeaways

- Robotics, a field with roots as far back as ancient Greece, is undergoing a period of explosive growth, driven by improvements in computational power, better sensors and new materials.
- Producing increasingly more autonomous machines requires new computational approaches to designing and fabricating functional machines at scale.
- MIT-CSAIL, in partnership with Toyota Motor Corporation, have set out to create a car that will never be responsible for a collision. This project builds on MIT's collaborative work with the National University of Singapore, which demonstrates safe self-driving vehicles at low speeds in low-complexity environments.

Let's start with your background.

I'm a roboticist. I started as a computer science and math major in college. Towards the end of my college career, I met John Hopcroft, who then became my Ph.D. thesis advisor. At one point, he delivered a very inspiring talk, in which he observed that many of the classical computer science problems have already been solved. Instead, it was now time for the grand applications of computer science, including, of course, robotics.

I continued on to Cornell, where I spent about five years working on my Ph.D. In those days, Cornell produced a lot of foundational work in robotics. Following my Ph.D., I accepted a job as an assistant professor at Dartmouth College, where I founded a robotics research lab. And in 2003, I made my way to MIT, where I work today. The main objective of my work is to advance the science of autonomy: how can machines operate without human input in the physical world? I'm especially interested in cases where multiple machines coordinate to accomplish something that neither machine is able to solve alone.

Can you briefly sketch out the origins of robotics as a field?

Early robotic concepts date far back. The ancient Greeks engineered complex machines out of simple mechanisms, which could open temple doors or play musical instruments. In the 18th century, Swiss watchmakers designed automata, hard-coded mechanisms that could play musical instruments, write, and even paint.

In the early 1960s, George Devol, who is considered the father of industrial robotics, built a robot called Unimate. His work marked a major leap over previous automata. This robot was programmable to perform disparate tasks: It could swing a golf club or pour wine. Later in the '70s, The Stanford Cart presented an early example of a robotic mobile device; it was, in effect, a mobile robot and the first of its kind to combine perception and action with planning. The robot took painstakingly long to traverse the trajectory from one end of a small parking lot to the other, yet its success marked the dawn of technological breakthroughs in robotics centered on machine perception, planning and learning.

Over the past decade, the field writ large experienced remarkable progress, driven by a number of important trends: computational power has been increasing at a breakneck pace, the hardware

required to interact with the physical world—the sensors and motors themselves—have become smaller and more reliable, and an array of new materials continue pushing the limits of design.

In parallel, the community has achieved breakthrough progress in the science and application of planning, control, and perception. Today's robots have an incredible set of skills: The ability to make maps, localize, as well as smart decision making and learning capacity. Up until recently, these twin threads of progress and knowledge have been pursued somewhat independently. We're now witnessing their convergence. The state-of-the-art in each field is coalescing to deliver results that were merely a dream even just ten years ago.

What is your primary focus in robotics?

My greatest interest is to advance the science of autonomy—in particular, systems that involve multiple robots working together. I want to make robots more capable and independent, and I want to see these advanced physical machines improve our lives.

A strong interplay underlies the robot's physical and software capabilities. In some cases, we need to invent new robot bodies to deliver on the capabilities that we want. In other cases, we repurpose existing robot bodies to do novel things.

I have been interested in how to make capable robots, faster. One such approach advances a universal robot cell, or module, that can be reused to make all kinds of different robots. This idea, in turn, suggests shape-shifting robots—modular, cellular robots with the ability to adapt their geometric structure to the task at hand, autonomously. If you build a robot designed solely for a single task, the robot will perform that task well, but will, by its very design, perform poorly on an unrelated or different task in a foreign environment. In contrast, if you design machines with the ability to contextually re-organize their internal geometry, you obtain the right robot body for the right application and environment.

An alternative solution to this problem is to make task-specific robots more quickly, by automating the design and fabrication of robots from high-level specifications. In other words, create a robot compiler. The general idea is to automatically convert user-defined functional specifications into physical one-of-a-kind machines that meet those specifications.

You've also spent time working on autonomous vehicles. Can you tell us about your work in that domain?

I have been working on self-driving cars as part of a collaboration between MIT and the National University of Singapore for several years now. We are developing a system of autonomous golf carts and autonomous electric vehicles for mobility on demand. This project is being run under the auspices of the Singapore-MIT Alliance for Research and Technology (SMART).

Through our work, we have already demonstrated that self-driving cars, at low speeds in low-complexity environments are in fact reliable! We are now extending these self-driving vehicles to an urban mobility system, similar in spirit to the now-popular shared bicycle programs. Bicycle programs, for their part, face some basic challenges. Over time, some stations become depleted of bikes while others overflow. Cities respond by hiring people and trucks to rebalance the vehicles in what amounts to a costly and inefficient exercise.

Imagine, instead, if the bikes could drive themselves to the next destination to match local demand. In our model, the self-driving car transports you to your destination and then coordinates with other cars to pick up the next person in line. The car then drives itself to the appropriate next spot. With sufficient investment, this idea has the potential to turn transportation into a utility, available to people in cities anywhere and anytime.

In addition, we recently launched a collaboration between MIT CSAIL and Toyota to develop a safe car that will never be responsible for a collision, becoming, over time, a trusted partner for the human driver. I am very, very excited about these new research directions.

What are some of the main challenges in making autonomous vehicles that can safely navigate cities?

The major challenges hinge on environmental complexity, speed of movement, weather, and human-interaction. The current machine perception and control algorithms are not smart enough nor fast enough to respond to the extreme driving circumstances we encounter in heavy congestion and bad weather. Imagine traffic in New Delhi, the Philippines, or L.A. It's treacherous for numerous reasons: congestion, erratic driver behavior, coordination through

silent hand gestures among human drivers, inclement weather, heavy rain, snow, poor visibility, and so on. The self-driving car, as a problem, is not solved. It's important to keep that in mind.

Can you walk us through the self-driving car hardware and software makeup?

The vehicles in the SMART project contain a minimalist hardware configuration. They use two forward-pointed laser scanners, one for mapping and localization and another for obstacle detection. They also carry a forward-pointing camera for detecting moving obstacles (e.g., a pedestrian), as well as side and rear-pointing laser scanners, an Inertial Measurement Unit (IMU), and wheel encoders.

What machine learning techniques are critical to the next stages of autonomous car development?

Deep learning is engendering a great deal of enthusiasm. Armed with the latest deep learning packages, we can begin to recognize objects in previously impossible ways. Machine learning presents an interesting challenge for driving because the car requires the utmost reliability and efficiency in how other cars, obstacles, and objects in the surrounding environment are recognized and taken into account. In other words, there is no room for error, especially as high speeds.

What's next—do cars take over driving altogether?

A bedrock assumption in our work with Toyota is that driving is fun. We'd like to see a partnership between the car and and its (human) driver. I would like for my car to learn and adapt to my preferences and normal state. In the future, the car might be able to determine that I'm having a difficult day based on how I speak and then keep a close eye on my driving; if I am about to make a mistake, for example, by miscalculating the speed of the incoming traffic during a left turn, the car could intercede and correct the mistake. This sort of override would operate in the same spirit of the anti-lock braking systems we have come to rely on.

I want to conclude by reminding everyone that an accident happens in the U.S. every 5 seconds. The costs, both in terms of human life, and in economic terms, are simply staggering. We envision that car of the future will possess a parallel autonomy system able to correct the mistakes of the human drivers and prevent those deadly colli-

sions. This car, in time, will learn a lot about its human operator by way of becoming a trusted partner, without taking away the joy of driving.

CHAPTER 7

Gurjeet Singh: Using Topology to Uncover the Shape of Your Data

Gurjeet Singh is CEO and co-founder of Ayasdi, a company that leverages machine intelligence software to automate and accelerate discovery of data insights. The author of numerous patents and publications in top mathematics and computer science journals, Gurjeet has developed key mathematical and machine learning algorithms for topological data analysis.

Key Takeaways

- The field of topology studies the mapping of one space into another through continuous deformations.
- Machine learning algorithms produce functional mappings from an input space to an output space and lend themselves to be understood using the formalisms of topology.
- A topological approach allows you to study datasets without assuming a shape beforehand and to combine various machine learning techniques while maintaining guarantees about the underlying shape of the data.

Let's get started by talking about your background and how you got to where you are today.

I am a mathematician and a computer scientist, originally from India. I got my start in the field at Texas Instruments, building integrated software and performing digital design. While at TI, I got to work on a project using clusters of specialized chips called digital signal processors (DSPs) to solve computationally hard math problems.

As an engineer by training, I had a visceral fear of advanced math. I didn't want to be found out as a fake, so I enrolled in the Computational Math program at Stanford. There, I was able to apply some of my DSP work to solving partial differential equations and demonstrate that a fluid dynamics researcher need not buy a supercomputer anymore. They could just employ a cluster of DSPs to run the system. I then spent some time in mechanical engineering building similar GPU-based partial differential equation solvers for mechanical systems. Finally, I worked in Andrew Ng's lab at Stanford, building a quadruped robot and programming it to learn to walk by itself.

Then one day I saw a note from my advisor, Gunnar Carlsson, describing how he was applying topology to explain real data sets. He explained how topology could be applied equally well to four or five very distinct and interesting problem areas. That was really exciting, and I started working with him on the topic. The project was an academic success and DARPA (the Defense Advanced Research Projects Agency) asked us to commercialize our research and start a company. That's how we started Ayasdi.

Can you tell us about the evolution of topology, broadly speaking, and share some insights as to why it is so useful for unifying disparate areas in machine intelligence?

Topology is a very old branch of mathematics. It was developed in the 1700s by mathematicians like Euler. It was originally developed to quantify the qualitative aspects of algebraic equations. For example, if you have the equation for a circle, topology is the area of math that allows you to say that, for example, "Oh, a circle is a single connected thing; it divides the plane into an inside and an outside; and it has a simple connectivity structure." Over the course of its development over the last 300 years, it has become the study of mapping one space into another.

For example, there are two large classes of machine learning algorithms. There are supervised machine learning algorithms and the unsupervised ones. Furthermore, within supervised algorithms, there are two types: algorithms that take an input vector to predict a number, and algorithms that take a vector to produce a class label.

On the unsupervised side, there are two distinct methods. What unifies these four distinct functions is they all produce functional mappings from an input space to an output space. The built-in formalism of topology allows you to learn across different types of functions. So if you want to combine the results of these various learning algorithms together, topology allows you to do that, while still maintaining guarantees about the underlying shape or distributions. That's the first critical insight.

The second insight is that, basically, all machine learning algorithms solve optimization problems. The machine learning algorithm assumes a particular shape of the data for each problem. Then the optimization procedure finds the best parameters that make the data look like that shape. Topology does the reverse. Topology, even though it utilizes all these machine learning algorithms under the covers, allows you to discover the underlying shape of the data so that you don't have to assume it.

What are some of the key concepts around the application of topology to machine learning?

It's very simple. There is only one key idea: data has shape, and shape has meaning. In standard machine learning, the shape of the data is usually an afterthought. Topology puts the shape front and center, i.e., as being the most important aspect of your data.

What are the real world applications of this technology? Why is this important?

Today, we're awash in data. Machine learning algorithms were developed as a methodology to extract value from increasingly large and complex datasets. However, there are now many algorithms from which to choose. The incomplete or incorrect application of machine learning algorithms can lead to missing or even erroneous conclusions.

Topology addresses this issue of increasing data complexity, by the comprehensive investigation of your dataset with any algorithm or

combination of algorithms, and presents an objective result (i.e., no information loss).

Using a topological approach, what does a typical investigation look like?

One huge benefit of using topology is that you don't have to presuppose a library of shapes. You don't have to say, "Okay, I know what a circle looks like. A circle is our prototype now." Topology represents your underlying data in a combinatorial form. It constructs a network in which every node in said network contains a subset of your data, and two nodes are connected to each other if they share some data.

If you think about it from a tabular perspective, you feed it your table, and the output is this graph representation in which every node is a subset of the rows. But a row can appear in more than one node, and whenever that happens you connect them. This very simple structure has a two huge advantages. The first is that irrespective of the underlying machine learning algorithms that have been combined in a particular investigation, the output will always look like this graph. The second is that this network form is very *computable*; you can easily build things on top of it, like recommender systems, piecewise linear models, gradient operators, and so on.

Can you generalize that to another example, where the shape is not necessarily a circle?

Imagine that you had the letter Y on graph paper, and you're sampling data from it. Clustering the raw data doesn't make sense, because you'll recover a single cluster—if you're lucky. If you want to build a regression on it, that's also wrong, because the data is nonlinear.

Imagine you use the centrality function to reduce the dimensions. So for every point on the Y, you measure the sum of its distance to every other point on the Y. The value of the function at the joining point in the middle of the Y will be low, because all those points are central. The tips of the Y will be high, because they're far from everything else. Now, if you merge your dimensionality reduction function with clustering, then in the low range you get a single cluster, because it's the middle of the Y. As you go out of that middle range, you start seeing three clusters, because those are the spokes of the Y.

Is it fair to generalize that when performing a topological investigation, the first order of business is using some form of dimensionality reduction algorithm?

That is correct. Once you reduce the data, compact it, and get the benefit of being cognizant of the topology, you're able to maintain the shape while uncovering relationships in and among the data. Basically, different dimensionality reduction algorithms will illuminate different aspects of the shape.

Is it just a matter of throwing all of these algorithms up against the wall to see what's interesting?

Yes, essentially, you throw as many of these functions at the data as you can. These are usually computationally expensive functions e.g., Isomap. Topology allows you to compare across them very smoothly. You can discover statistically significant differences in your data in an algorithmic way. The machinery allows you to do that very beautifully.

Given you can map the same data into different views/representations, is there a single view that's analytically superior towards the goal of understanding any particular problem?

You have to be careful. You don't necessarily want a single view. There is no single right answer, because different combinations of these algorithms will produce different types of insights in your data. They are all equally valid if you can prove their statistical validity. You don't want to somehow confine yourself to a single right answer. You want to pull out statistically significant ideas from all of these.

Do the insights or results across different views of the same data ever contradict each other?

In fact, one of the things that's beneficial in our approach is that these algorithms are correlated with each other. In many cases, you find the evidence for the same phenomena over and over again across multiple maps. Wherever that happens, you can be more confident about what you found.

So in other words, features that persist across different views or mappings are more significant?

One of the areas of topology that is especially interesting to this discussion is homology. Persistent homology essentially talks about the stability of features that you discover using topological methods. You can imagine in many machine learning settings, you have these algorithms that are parameterized in various ways. You somehow have to say, "Okay, this is the set of parameters that I'm going to choose." You can imagine in all of those settings, it's very helpful to have tools that tell you the stability range of these parameters. That across this or that range they are going to be stable.

Imagine if you stare at a circle from a long distance, from far enough away, you might conclude that a circle is just a dot. So you have to ask, "Over what range of distances do I call a circle a circle?" And this generalizes to other shapes and the various resolutions in which they can be viewed. There's a really interesting body of research around this. In fact, some parts of this work are also used at Ayasdi (in our code base), but we don't expose it.

Looking ahead, what would you consider the most exciting developments in machine intelligence? Is persistent homology the kind of the thing you would tell folks to look at, either inside or outside of topology?

This is the golden age of machine learning. There is so much interesting work going on. We've turned a corner; in the past, people working in the field tended to be married to a particular method. Now, all of a sudden, people are open to new things. For example, all through 1980s there was a focus on logistic regression, and nobody wanted to do anything else. By the 2000s, the focus has shifted to support vector machines (SVMs) and, again, nobody wanted to do anything else. These days, the whole field seems to have matured. Everybody is open to different points of view.

I think there's a lot of interesting work going on in feature engineering. That's interesting, because on the one hand, we have this whole deep learning core process. So some will tell you that we don't need feature engineering. But on the other hand, everybody who does feature engineering *with* deep learning produces much better results.

In topology more specifically, the exciting news is that we now have a few things that work. And we are on the cusp of attaining a theoretical understanding of why that happens; that is, why the things that work—work. When we understand that, we can begin to evolve it. These are indeed exciting times!

CHAPTER 8

Ilya Sutskever: Unsupervised Learning, Attention, and Other Mysteries

Ilya Sutskever is a research scientist at Google and the author of numerous publications on neural networks and related topics. Sutskever is a co-founder of DNNresearch and was named Canada's first Google Fellow.

Key Takeaways

- Since humans can solve perception problems very quickly, despite our neurons being relatively slow, moderately deep and large neural networks have enabled machines to succeed in a similar fashion.
- Unsupervised learning is still a mystery, but a full understanding of that domain has the potential to fundamentally transform the field of machine learning.
- Attention models represent a promising direction for powerful learning algorithms that require ever less data to be successful on harder problems.

Let's start with your background. What was the evolution of your interest in machine learning, and how did you zero in on your Ph.D. work?

I started my Ph.D. just before deep learning became a thing. I was working on a number of different projects, mostly centered around neural networks. My understanding of the field crystallized when collaborating with James Martins on the Hessian-free optimizer. At the time, greedy layer-wise training (training one layer at a time) was extremely popular. Working on the Hessian-free optimizer helped me understand that if you just train a very *large* and *deep* neural network on a lot of data, you will almost necessarily succeed.

Taking a step back, when solving naturally occurring machine learning problems, you use some model. The fundamental question is whether you believe that this model can solve the problem for some setting of its parameters? If the answer is no, then the model will not get great results, no matter how good its learning algorithm. If the answer is yes, then it's only a matter of getting the data and training it. And this is, in some sense, the primary question. Can the model represent a good solution to the problem?

There is a compelling argument that large, deep neural networks should be able to represent very good solutions to perception problems. It goes like this: human neurons are slow, and yet humans can solve perception problems extremely quickly and accurately. If humans can solve useful problems in a fraction of a second, then you should only need a very small number of massively-parallel steps in order to solve problems like vision and speech recognition. This is an old argument—I've seen a paper on this from the early eighties.

This suggests that if you train a large, deep neural network with ten or 15 layers, on something like vision, then you could basically solve it. Motivated by this belief, I worked with Alex Krizhevsky towards demonstrating it. Alex had written an extremely fast implementation of 2D convolutions on a GPU, at a time when few people knew how to code for GPUs. We were able to train neural networks larger than ever before and achieve much better results than anyone else at the time.

Nowadays, everybody knows that if you want to solve a problem, you just need to get a lot of data and train a big neural net. You might not solve it perfectly, but you can definitely solve it better than you could have possibly solved it without deep learning.

Not to trivialize what you're saying, but you say throw a lot of data at a highly parallel system, and you'll basically figure out what you need?

Yes, but: although the system is highly parallel, it is its sequential nature that gives you the power. It's true we use parallel systems because that's the only way to make it fast and large. But if you think of what depth represents—depth is the sequential part.

And if you look at our networks, you will see that each year they are getting deeper. It's amazing to me that these very vague, intuitive arguments turned out to correspond to what is actually happening. Each year the networks that do best in vision are deeper than they were before. Now we have twenty-five layer computational steps, or even more depending on how you count.

What are the open problems, theoretically, in making deep learning as successful as it can be?

The huge open problem would be to figure out how you can do more with less data. How do you make this method less data-hungry? How can you input the same amount of data, but better formed?

This ties in with the one of greatest open problems in machine learning—unsupervised learning. How do you even think about unsupervised learning? How do you benefit from it? Once our understanding improves and unsupervised learning advances, this is where we will acquire new ideas, and see a completely unimaginable explosion of new applications.

What's our current understanding of unsupervised learning? And how is it limited in your view?

Unsupervised learning is mysterious. Compare it to supervised learning. We know why supervised learning works. You have a big model, and you're using a lot of data to define the cost—the training error—which you minimize. If you have a lot of data, your training error will be close to your test error. Eventually, you get to a low test error, which is what you wanted from the start.

But I can't even articulate what it is we want from unsupervised learning. You want something; you want the model to *understand*...whatever that means. Although we currently understand very little about unsupervised learning, I am also convinced that the explanation is right under our noses.

Are you aware of any promising avenues that people are exploring towards a deeper, conceptual understanding of why unsupervised learning does what it does?

There are plenty of people trying various ideas, mostly related to density modeling or generative models. If you ask any practitioner how to solve a particular problem, they will tell you to get the data and apply supervised learning. There is not yet an important application where unsupervised learning makes a profound difference.

Do we have any sense of what success means? Even a rough measure of how well an unsupervised model performs?

Unsupervised learning is always a means for some other end. In supervised learning, the learning itself is what you care about. You've got your cost function, which you want to minimize. In unsupervised learning, the goal is always to help some other task, like classification or categorization. For example, I might ask a computer system to passively watch a lot of YouTube videos (so unsupervised learning happens here), then ask it to recognize objects with great accuracy (that's the final supervised learning task).

Successful unsupervised learning enables the subsequent supervised learning algorithm to recognize objects with accuracy that would not be possible without the use of unsupervised learning. It's a very measurable, very visible notion of success. And we haven't achieved it yet.

What are some other areas where you see exciting progress?

A general direction which I believe to be extremely important are learning models capable of more sequential computations. I mentioned how I think that deep learning is successful because it can do more sequential computations than previous ("shallow") models. And so models that can do even more sequential computation should be even more successful because they are able to express more intricate algorithms. It's like allowing your parallel computer to run for more steps. We already see the beginning of this, in the form of attention models.

And how do attention models differ from the current approach?

In the current approach, you take your input vector and give it to the neural network. The neural network runs it, applies several processing stages to it, and then gets an output. In an attention model,

you have a neural network, but you run the neural network for much longer. There is a mechanism in the neural network, which decides which part of the input it wants to "look" at. Normally, if the input is very large, you need a large neural network to process it. But if you have an attention model, you can decide on the best size of the neural network, independent of the size of the input.

So then how do you decide where to focus this attention in the network?

Say you have a sentence, a sequence of say, 100 words. The attention model will issue a query on the input sentence and create a distribution over the input words, such that a word which is more similar to the query will have higher probability, and words which are less similar to the query will have lower probability. Then you take the weighted average of them. Since every step is differentiable, we can train the attention model where to look with backpropagation, which is the reason for its appeal and success.

What kind of changes do you need to make to the framework itself? What new code do you need to insert this notion of attention?

Well, the great thing about attention, at least differentiable attention, is that you don't need to insert any new code to the framework. As long as your framework supports element-wise multiplication of matrices or vectors, and exponentials, that's all you need.

So attention models address the question you asked earlier: how do we make better use of existing power with less data?

That's basically correct. There are many reasons to be excited about attention. One of them is that attention models simply work better, allowing us to achieve better results with less data. Also bear in mind that humans clearly have attention. It is something that enables us to get results. It's not just an academic concept. If you imagine a really smart system, surely, it too will have attention.

What are some of the key issues around attention?

Differentiable attention is computationally expensive because it requires accessing your entire input at each step of the model's operation. And this is fine when the input is a sentence that's only, say, 100 words, but it's not practical when the input is a ten-thousand word document. So one of the main issues is speed. Attention should be fast, but differentiable attention is not fast. Reinforcement learning of attention is potentially faster, but training attentional

control using reinforcement learning over thousands of objects would be non-trivial.

Is there an analog, in the brain, as far as we know, for unsupervised learning?

The brain is a great source of inspiration, if looked at correctly. The question of whether the brain does unsupervised learning or not depends to some extent on what you consider to be unsupervised learning. In my opinion, the answer is unquestionably yes. Look at how people behave, and notice that people are not really using supervised learning at all. Humans never use any supervision of any kind. You start reading a book, and you understand it, and all of a sudden you can do new things that you couldn't do before. Consider a child, sitting in class. It's not like the student is given lots of input/output examples. The supervision is extremely indirect; so there's necessarily a lot of unsupervised learning going on.

Your work was inspired by the human brain and its power. How far does the neuroscientific understanding of the brain extend into the realm of theorizing and applying machine learning?

There is a lot of value of looking at the brain, but it has to be done carefully, and at the right level of abstraction. For example, our neural networks have units which have connections between them, and the idea of using slow interconnected processors was directly inspired by the brain. But it is a faint analogy.

Neural networks are designed to be computationally efficient in software implementations rather than biologically plausible. But the overall idea was inspired by the brain, and was successful. For example, convolutional neural networks echo our understanding that neurons in the visual cortex have very localized perceptive fields. This is something that was known about the brain, and this information has been successfully carried over to our models. Overall, I think that there is value in studying the brain, if done carefully and responsibly.

CHAPTER 9

Oriol Vinyals: Sequence-to-Sequence Machine Learning

Oriol Vinyals is a research scientist at Google working on the DeepMind team by way of previous work with the Google Brain team. He holds a Ph.D. in EECS from University of California, Berkeley, and a Master's degree from University of California, San Diego.

Key Takeaways

- Sequence-to-sequence learning using neural networks has delivered state of the art performance in areas such as machine translation.
- While powerful, such approaches are constrained by a number of factors, including computational ones. LSTMs have gone a long way towards pushing the field forward.
- Besides image and text understanding, deep learning models can be taught to "code" solutions to a number of well-known algorithmic challenges, including the Traveling Salesman Problem.

Let's start with your background.

I'm originally from Barcelona, Spain, where I completed my undergraduate studies in both mathematics and telecommunication engineering. Early on, I knew I wanted to study AI in the U.S. I spent

nine months at Carnegie Mellon, where I finished my undergraduate thesis. Afterward, I received my Master's degree at UC San Diego before moving to Berkeley for my Ph.D. in 2009.

While interning at Google during my Ph.D., I met and worked with Geoffrey Hinton, which catalyzed my current interest in deep learning. By then, and as a result of wonderful internship experiences at both Microsoft and Google, I was determined to work in industry. In 2013, I joined Google full time. My initial research interest in speech recognition and optimization (with an emphasis on natural language processing and understanding) gave way to my current focus on solving these and other problems with deep learning, including most recently, generating learning algorithms from data.

Tell me about your change in focus as you moved away from speech recognition. What are the areas that excite you the most now?

My speech background inspired my interest in sequences. Most recently, Ilya Sutskever, Quoc Le and I published a paper on mapping from sequences-to-sequences so as to enable machine translation from French to English using a recurrent neural net.

For context, supervised learning has demonstrated success in cases where the inputs and outputs are vectors, features or classes. An image fed into these classical models, for example, will output the associated class label. Until quite recently, we have not been able to feed an image into a model and output a sequence of words that describe said image. The rapid progress currently underway can be traced to the availability of high quality datasets with image descriptions (MS COCO), and in parallel, to the resurgence of recurrent neural networks.

Our work recast the machine translation problem in terms of sequence-based deep learning. The results demonstrated that deep learning can map a sequence of words in English to a corresponding sequence of words in Spanish. By virtue of deep learning's surprising power, we were able to wrangle state-of-the-art performance in the field rather quickly. These results alone suggest interesting new applications, for example, automatically distilling a video into four descriptive sentences.

Where does the sequence-to-sequence approach not work well?

Suppose you want to translate a single sentence of English to its French analog. You might use a large corpus of political speeches

and debates as training data. A successful implementation could then convert political speech into any number of languages. You start to run into trouble though when you attempt to translate a sentence from, say, Shakespearean English, into French. This domain shift strains the deep learning approach, whereas classical machine translation systems use rules that make them resilient to such a shift.

Further complicating matters, we lack the computational resources to work on sequences beyond a certain length. Current models can match sequences of length 200 with corresponding sequences of length 200. As these sequences elongate, longer runtimes follow in tow. While we're currently constrained to a relatively small universe of documents, I believe we'll see this limit inevitably relax over time. Just as GPUs have compressed the turnaround time for large and complex models, increased memory and computational capacity will drive ever longer sequences.

Besides computational bottlenecks, longer sequences suggest interesting mathematical questions. Some years ago, Hochreiter introduced the concept of a vanishing gradient. As you read through thousands of words, you can easily forget information that you read three thousand words ago; with no memory of a key plot turn in chapter three, the conclusion loses its meaning. In effect, the challenge is memorization. Recurrent neural nets can typically memorize 10–15 words. But if you multiply a matrix fifteen times, the outputs shrink to zero. In other words, the gradient vanishes along with any chance of learning.

One notable solution to this problem relies on Long Short Term Memory (LSTMs). This structure offers a smart modification to recurrent neural nets, empowering them to memorize far in excess of their normal limits. I've seen LSTMs extend as far as 300–400 words. While sizable, such an increase is only the start of a long journey toward neural networks that can negotiate text of everyday scale.

Taking a step back, we've seen several models emerge over the last few years that address the notion of memory. I've personally experimented with the concept of adding such memory to neural networks: Instead of cramming everything into a recurrent net's hidden state, memories let you recall previously seen words towards the goal of optimizing the task at hand. Despite incredible progress in recent years, the deeper, underlying challenge of what it means to

represent knowledge remains, in itself, an open question. Nevertheless, I believe we'll see great progress along these lines in the coming years.

Let's shift gears to your work on producing algorithms. Can you share some background on the history of those efforts and their motivation?

A classic exercise in demonstrating the power of supervised learning involves separating some set of given points into disparate classes: this is class A; this is class B, etc. The XOR (the "exclusive or" logical connective) problem is particularly instructive. The goal is to "learn" the XOR operation, i.e., given two input bits, learn what the output should be. To be precise, this involves two bits and thus four examples: 00, 01, 10 and 11. Given these examples, the output should be: 0, 1, 1 and 0. This problem isn't separable in a way that a linear model could resolve, yet deep learning matches the task. Despite this, currently, limits to computational capacity preclude more complicated problems.

Recently, Wojciech Zaremba (an intern in our group) published a paper entitled "Learning to Execute," which described a mapping from python programs to the result of executing those same programs using a recurrent neural network. The model could, as a result, predict the output of programs written in python merely by reading the actual code. This problem, while simply-posed, offered a good starting point. So, I directed our attention to an NP-hard problem.

The algorithm in question is a highly complex and resource-intensive approach to finding exactly the shortest path through all the points in the famous Traveling Salesman Problem. Since its formulation, this problem has attracted numerous solutions that use creative heuristics while trading off between efficiency and approximation. In our case, we investigated whether deep learning system could infer useful heuristics on par with existing literature using the training data alone.

For efficiency's sake, we scaled down to ten cities, rather than the more common 10,000 or 100,000. Our training set input city locations and output the shortest paths. That's it. We didn't want to expose the network to any other assumptions about the underlying problem.

A successful neural net should be able to recover the behavior of finding a way to traverse all given points to minimize distance. Indeed, in a rather magical moment, we realized it worked.

The outputs, I should note, might be slightly sub-optimal because this is, after all, probabilistic in nature: But it's a good start. We hope to apply this method a range of new problems. The goal is not to rip and replace existing, hand-coded solutions. Rather, our effort is limited to replacing heuristics with machine learning.

Will this approach eventually make us better programmers?

Consider coding competitions. They kick off with a problem statement written in plain English: "In this program, you will have to find A, B and C, given assumptions X, Y and Z." You then code your solution and test it on a server. Instead, imagine for a moment a neural network that could read a such a problem statement in natural language and afterwards learn an algorithm that at least approximates the solution, and even perhaps returns it exactly. This scenario may sound far-fetched. Bear in mind though, just a few years ago, reading python code and outputting an answer that approximates what the code returns sounded quite implausible.

What do you see happening with your work over the next five years? Where are the greatest unsolved problems?

Perhaps five years is pushing it, but the notion of a machine reading a book for comprehension is not too distant. In a similar vein, we should expect to see machines that answer questions by learning from the data, rather than following given rule sets. Right now, if I ask you a question, you go to Google and begin your search; after some number of iterations, you might return with an answer. Just like you, machines should be able to run down an answer in response to some question. We already have models that move us in this direction on very tight data sets. The challenges going forward are deep: How do you distinguish correct and incorrect answers? How do you quantify wrongness or rightness? These and other important questions will determine the course of future research.

CHAPTER 10

Reza Zadeh: On the Evolution of Machine Learning

Reza Zadeh is a consulting professor at the Institute for Computational and Mathematical Engineering at Stanford University and a technical advisor to Databricks. His work focuses on machine learning theory and applications, distributed computing, and discrete applied mathematics.

Key Takeaways

- Neural networks have made a comeback and are playing a growing role in new approaches to machine learning.
- The greatest successes are being achieved via a supervised approach leveraging established algorithms.
- Spark is an especially well-suited environment for distributed machine learning.

Tell us a bit about your work at Stanford.

At Stanford, I designed and teach distributed algorithms and optimization (CME 323) as well as a course called discrete mathematics and algorithms (CME 305). In the discrete mathematics course, I teach algorithms from a completely theoretical perspective, meaning that it is not tied to any programming language or framework, and we fill up whiteboards with many theorems and their proofs.

On the more practical side, in the distributed algorithms class, we work with the Spark cluster programming environment. I spend at least half my time on Spark. So all the theory that I teach in regard to distributed algorithms and machine learning gets implemented and made concrete by Spark, and then put in the hands of thousands of industry and academic folks who use commodity clusters.

I started running MapReduce jobs at Google back in 2006, before Hadoop was really popular or even known; but MapReduce was already mature at Google. I was 18 at the time, and even then I could see clearly that this is something that the world needs outside of Google. So I spent a lot of time building and thinking about algorithms on top of MapReduce, and always worked to stay current, long after leaving Google. When Spark came along, it was nice that it was open-source and one could see its internals, and contribute to it. I felt like it was the right time to jump on board because the idea of an RDD was the right abstraction for much of distributed computing.

From your time at Google up to the present work you're doing with Spark, you have had the chance to see some of the evolution of machine learning as it ties to distributed computing. Can you describe that evolution?

Machine learning has been through several transition periods starting in the mid-90s. From 1995–2005, there was a lot of focus on natural language, search, and information retrieval. The machine learning tools were simpler than what we're using today; they include things like logistic regression, SVMs (support vector machines), kernels with SVMs, and PageRank. Google became immensely successful using these technologies, building major success stories like Google News and the Gmail spam classifier using easy-to-distribute algorithms for ranking and text classification—using technologies that were already mature by the mid-90s.

Then around 2005, neural networks started making a comeback. Neural networks are a technology from the 80s—some would even date them back to the 60s—and they've become "retrocool" thanks to their important recent advances in computer vision. Computer vision makes very productive use of (convolutional) neural networks. As that fact has become better established, neural networks are making their way into other applications, creeping into areas like natural language processing and machine translation.

But there's a problem: neural networks are probably the most challenging of all the mentioned models to distribute. Those earlier models have all had their training successfully distributed. We can use 100 machines and train a logistic regression or SVM without much hassle. But developing a distributed neural network learning setup has been more difficult.

So guess who's done it successfully? The only organization so far is Google; they are the pioneers, yet again. It's very much like the scene back in 2005 when Google published the MapReduce paper, and everyone scrambled to build the same infrastructure. Google managed to distribute neural networks, get more bang for their buck, and now everyone is wishing they were in the same situation. But they're not.

Why is an SVM or logistic regression easier to distribute than a neural network?

First of all, evaluating an SVM is a lot easier. After you've learned an SVM model or logistic regression model—or any linear model—the actual evaluation is very fast. Say you built a spam classifier. A new email comes along; to classify it as spam or not it takes very little time, because it's just one dot product (in linear algebra terms). When it comes to a neural network, you have to do a lot more computation—even after you have learned the model—to figure out the model's output. And that's not even the biggest problem. A typical SVM might be happy with just a million parameters, but the smallest successful neural networks I've seen have around 6 million—and that's the absolutely smallest. Another problem is that the training algorithms don't benefit from much of optimization theory. Most of the linear models that we use have mathematical guarantees on when training is finished. They can guarantee when you have found the best model you're going to find. But the optimization algorithms that exist for neural networks don't afford such guarantees. You don't know after you've trained a neural network whether, given your setup, this is the best model you could have found. So you're left wondering if you would have a better model if you kept on training.

As neural networks become more powerful, do you see them subsuming more and more of the work that used to be the bread and butter of linear methods?

I think so, yes. Actually that's happening right now. There's always this issue that linear models can only discriminate linearly. In order to get non-linearities involved, you would have to add or change features, which involves a lot of work. For example, computer vision scientists spent a decade developing and tuning these things called SIFT features, which enable image classification and other vision tasks using linear methods. But then neural networks came along and SIFT features became unnecessary; the neural network approach is to make features automatically as part of the training.

But I think it's asking for too much to say neural networks can replace all feature construction techniques. I don't think that will happen. There will always be a place for linear models and good human-driven feature engineering. Having said that, pretty much any researcher who has been to the NIPS Conference is beginning to evaluate neural networks for their application. Everyone is testing whether their application can benefit from the non-linearities that neural networks bring.

It's not like we never had nonlinear models before. We have had them—many of them. It's just that the neural network model happens to be particularly powerful. It can really work for some applications, and so it's worth trying. That's what a lot of people are doing. And when they see successes, they write papers about them. So far, I've seen successes in speech recognition, in computer vision, and in machine translation. It is a very wide array of difficult tasks, so there is good reason to be excited.

Why is a neural network so powerful compared to the traditional linear and nonlinear methods that have existed up until now?

When you have a linear model, every feature is either going to hurt or help whatever you are trying to score. That's the assumption inherent in linear models. So the model might determine that if the feature is large, then it's indicative of class 1; but if it's small, it's indicative of class 2. Even if you go all the way up to very large values of the feature, or down to very small values of the feature, you will never have a situation where you say, "In this interval, the feature is indicative of class 1; but in another interval it's indicative of class 2."

That's too limited. Say you are analyzing images, looking for pictures of dogs. It might be that only a certain subset of a feature's values indicate whether it is a picture of a dog, and the rest of the values for that pixel, or for that patch of an image, indicate another class. You

can't draw a line to define such a complex set of relationships. Non-linear models are much more powerful, but at the same time they're much more difficult to train. Once again, you run into those hard problems from optimization theory. That's why for a long while we thought that neural networks weren't good enough, because they would over-fit, or they were too powerful. We couldn't do precise, guaranteed optimization on them. That's why they (temporarily) vanished from the scene.

Within neural network theory, there are multiple branches and approaches to computer learning. Can you summarize some of the key approaches?

By far the most successful approach has been a supervised approach where an older algorithm, called backpropagation, is used to build a neural network that has many different outputs.

Let's look at a neural network construction that has become very popular, called convolutional neural networks. The idea is that the machine learning researcher builds a model constructed of several layers, each of which handles connections from the previous layer in a different way.

In the first layer, you have a window that slides a patch across an image, which becomes the input for that layer. This is called a convolutional layer because the patch "convolves", it overlaps with itself. Then several different types of layers follow. Each have different properties, and pretty much all of them introduce nonlinearities.

The last layer has 10,000 potential neuron outputs; each one of those activations correspond to a particular label which identifies the image. The first class might be a cat; the second class might be a car; and so on for all the 10,000 classes that ImageNet has. If the first neuron is firing the most out of the 10,000 then the input is identified as belonging to the first class, a cat.

The drawback of the supervised approach is that you must apply labels to images while training. This is a car, this is a zoo, etc.

Right. And the unsupervised approach?

A less popular approach involves "autoencoders", which are unsupervised neural networks. Here the neural network is not used to classify the image, but to compress it. You read the image in the same way I just described, by identifying a patch and feeding the pixels into a convolutional layer. Several other layers then follow, including a middle layer which is very small compared to the others. It has relatively few neurons. Basically you're reading the image, going through a bottleneck, and then coming out the other side and trying to reconstruct the image.

No labels are required for this training, because all you are doing is putting the image at both ends of the neural network and training the network to make the image fit, especially in the middle layer. Once you do that, you are in possession of a neural network that knows how to compress images. And it's effectively giving you features that you can use in other classifiers. So if you have only a little bit of labeled training data, no problem—you always have a lot of images. Think of these images as non-labeled training data. You can use images to build an autoencoder, then from the autoencoder pull out features that are a good fit using a little bit of training data to find the neurons in your autoencoded neural network that are susceptible to particular patterns.

What got you into Spark? And where do you see that set of technologies heading?

I've known Matei Zaharia, the creator of Spark, since we were both undergraduates at Waterloo. And we actually interned at Google at the same time. He was working on developer productivity tools, completely unrelated to big data. He worked at Google and never touched MapReduce, which was my focus—kind of funny given where he ended up.

Then Matei went to Facebook, where he worked on Hadoop and became immensely successful. During that time, I kept thinking about distributing machine learning and none of the frameworks that were coming out—including Hadoop—looked exciting enough for me to build on top of because I knew from my time at Google what was really possible.

Tell us a bit about what Spark is, how it works, and why it's particularly useful for distributed machine learning.

Spark is a cluster computing environment that gives you a distributed vector that works similar to the vectors you're used to programming with on a single machine. You can't do everything you could with a regular vector; for example, you don't have arbitrary random access via indices. But you can, for example, intersect two vectors; you can union; you can sort. You can do many things that you would expect from a regular vector.

One reason Spark makes machine learning easy is that it works by keeping some important parts of the data in memory as much as possible without writing to disk. In a distributed environment, a typical way to get fault resilience is to write to disk, to replicate a disk across the network three times using HDFS.

What makes this suitable for machine learning is that the data can come into memory and stay there. If it doesn't fit in memory, that's fine too. It will get paged on and off a disk as needed. But the point is while it can fit in memory, it will stay there. This benefits any process that will go through the data many times—and that's most of machine learning. Almost every machine learning algorithm needs to go through the data tens, if not hundreds, of times.

Where do you see Spark vis-a-vis MapReduce? Is there a place for both of them for different kinds of workloads and jobs?

To be clear, Hadoop as an ecosystem is going to thrive and be around for a long time. I don't think the same is true for the MapReduce component of the Hadoop ecosystem.

With regard to MapReduce, to answer your question, no, I don't think so. I honestly think that if you're starting a new workload, it makes no sense to start in MapReduce unless you have an existing code base that you need to maintain. Other than that, there's no reason. It's kind of a silly thing to do MapReduce these days: it's the difference between assembly and C++. It doesn't make sense to write assembly code if you can write C++ code.

Where is Spark headed?

Spark itself is pretty stable right now. The biggest changes and improvements that are happening right now and happening in the next couple years are in the libraries. The machine learning library, the graph processing library, the SQL library, and the streaming libraries are all being rapidly developed, and every single one of them has an exciting roadmap for the next two years at least. These

are all features that I want, and it's very nice to see that they can be easily implemented. I'm also excited about community-driven contributions that aren't general enough to put into Spark itself, but that support Spark as a community-driven set of packages. I think those will also be very helpful to the long-tail of users.

Over time, I think Spark will become the de facto distribution engine on which we can build machine learning algorithms, especially at scale.

About the Author

David Beyer is an investor with Amplify Partners, an early-stage VC fund focused on the next generation of infrastructure IT, data, and information security companies. He began his career in technology as the co-founder and CEO of Chartio, a pioneering provider of cloud-based data visualization and analytics. He was subsequently part of the founding team at Patients Know Best, one of the world's leading cloud-based personal health record companies.