UNIVERSITY OF CALIFORNIA SAN DIEGO

Memcomputing Artificial Intelligence: Improving Learning Algorithms with Algorithms that Learn

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Physics (with a specialization in Computational Science)

by

Haik Manukian

Committee in charge:

Professor Massimiliano Di Ventra, Chair
Professor Gert Cauwenberghs
Professor Micheal Holst
Professor David Kleinfeld
Professor Brad Werner

2021

The dissertation of Haik Manukian is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

DEDICATION

To my family, who have shown me unconditional love and support throughout my life. To my friends, who somehow manage to tolerate my foolishness and in return fill my life with joy, humor, and sincere companionship. And of course to my Talin, whose smile reminds me everyday that I am among the luckiest people who have ever lived.

EPIGRAPH

Science, as well as technology, will in the near and in the farther future increasingly turn
from problems of intensity, substance, and energy,
to problems of structure, organization, information, and control.
—John von Neumann

The truth is a trap:
you can not get it without it getting you;
you cannot get the truth by capturing it, only by its capturing you.
– Søren Kierkegaard

Think what has never been thought before about what you see every day.
– Erwin Schrödinger

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

A multitude of factors came together to make this thesis possible, I've but put the cherry on top. My advisor and committee chair, Prof. Massimiliano Di Ventra, deserves the biggest share of thanks, for encouraging my learning and keeping my research on track throughout my years here. Also invaluable was the mentorship of Fabio Traversa, whose guidance was crucial during the initial phases of my research. Others in my research group, especially Forrest Sheldon, Rudy Pei, and Sean Bearden, have become great friends in addition to being academic companions. A huge heap of thanks also goes to Professors Enrico Ramirez-Ruiz and Greg Laughlin, who both played an integral role in nurturing my passions and abilities in scientific research during my undergraduate years. I'd also like to extend a special thanks to my dear friend, Ed, whose thoughtful dialogue over the years about computation, physics, and beyond, was a constant source of inspiration and helped clarify and develop many ideas I encountered during my research.

Chapter 2, in full, is a reprint of the material as it appears in, Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. "Memcomputing numerical inversion with self-organizing logic gates." IEEE transactions on neural networks and learning systems 29.6 (2017): 2645-2650. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in, Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. "Accelerating deep learning with memcomputing." Neural Networks 110 (2019): 1-7. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in, Manukian, Haik, Yan Ru Pei, Sean RB Bearden, and Massimiliano Di Ventra. "Mode-assisted unsupervised learning of restricted Boltzmann machines." Communications Physics 3, no. 1 (2020): 1-8. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material which was submitted for publication as, Manukian, Haik; Di Ventra, Massimiliano. Mode-assisted joint training of deep Boltzmann

machines. The dissertation author was the primary investigator and author of this paper.

VITA

2012	B. S. in Physics (Astrophysics) & B. A. in Mathematics, University of California Santa Cruz
2013-2015	M. A. in Applied Mathematics, University of California San Diego
2016-2021	Ph. D. in Physics with a Specialization in Computational Science, University of California San Diego

PUBLICATIONS

H Manukian, and M Di Ventra, "Mode-Assisted Joint Training of Deep Boltzmann Machines." *arXiv preprint arXiv:2102.08562*, 2021.

H Manukian, YR Pei, SR Bearden and M Di Ventra, "Mode-assisted unsupervised learning of restricted Boltzmann machines", *Communications Physics*, 2020

Pei, Yan Ru, Haik Manukian, and Massimiliano Di Ventra. "Generating Weighted MAX-2-SAT Instances with Frustrated Loops: an RBM Case Study." Journal of Machine Learning Research 21.159 (2020): 1-55.

Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. "Accelerating deep learning with memcomputing." Neural Networks 110 (2019): 1-7.

Bearden, S. R., Manukian, H., Traversa, F. L., and Di Ventra, M. (2018). Instantons in self-organizing logic gates. Physical Review Applied, 9(3), 034029.

Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. "Memcomputing numerical inversion with self-organizing logic gates." IEEE transactions on neural networks and learning systems 29.6 (2017): 2645-2650.

ABSTRACT OF THE DISSERTATION

Memcomputing Artificial Intelligence: Improving Learning Algorithms with Algorithms that Learn

by

Haik Manukian

Doctor of Philosophy in Physics (with a specialization in Computational Science)

University of California San Diego, 2021

Professor Massimiliano Di Ventra, Chair

Like sentinels guarding a secret treasure, computationally difficult problems define the edge of what can be accomplished across science and industry. The current computing orthodoxy, based on physical realization of the Turing machine via the von Neumann architecture, has begun to stagnate, as seen in the slowing down of Moore's 'law'. Hardware has saturated around its architectural (von Neumann) bottleneck, leading to a growing interest in unconventional computing architectures like quantum computing and neuromorphic computing. In this thesis, we examine a new alternative, the brain-inspired computing paradigm, memcomputing, which computes with and in memory. The dynamical systems induced by digital memcomputing

machines (DMMs) are applicable to a large family of optimization problems. DMMs use memory, or learn, from past dynamics to explore the phase space of the underlying problem in a drastically different way than typical algorithmic approaches, opening up many potential benefits from optimization to sampling.

Some of the most challenging computational problems (and biggest rewards) come from the attempt to instill intelligent behavior in machines, known as the field of artificial intelligence. The first chapter is a warm up with a hardware model of DMMs applied to numerical inversion problems, then we quickly move to the major theme throughout this work, which is the application of DMMs to computational demanding problems in artificial intelligence. The model we focus on is the Boltzmann machine, which is seen as an ancestor to the more popular feedforward neural networks. It no longer makes headlines not because it lacks capability, but rather our training methods lack the ability to train it, leaving much of its capacity unexplored. In this work we apply memcomputing to the training of Boltzmann machines through the development of mode-assisted training. This is a technique that uses memcomputing to sample the mode of the Boltzmann machine, and uses that to stabilize and improve the weight update algorithms. Mode-assisted training improves the performance of DBNs in a downstream supervised task, the unsupervised learning of RBMs as well as the joint training of deep Boltzmann machines, doing as well or better than networks containing two orders of magnitude more parameters.

Chapter 1

Introduction

This introduction serves as a guide through the research done during my time in Professor Di Ventra's group at UC San Diego. The unifying theme of this work is the application of the brain-inspired computing paradigm, memcomputing, to computationally demanding problems, primarily in artificial intelligence.

This introduction develops the two specific problems on the computational frontier addressed by this thesis through practical realizations of the memcomputing idea called Self-Organizing Logic Circuits (SOLCs). SOLCs are hardware level designs of digital memcomputing machines constructed using resistors, capacitors, voltage generators and memristors (dynamical resistors with memory). A simplified construction of SOLCs is developed and subsequently applied in Chapter 2 toward floating point numerical inversion.

Boltzmann machines are then introduced, which are an influential model in artificial intelligence (particularly in deep learning) based on the Ising model for ferromagnetism. The limitations of current Gibbs sampling approaches to their training are discussed, and mode-assisted training is introduced, which aids training with samples of the ground state of the Boltzmann machine. Finding the ground state of the Boltzmann machine is a difficult optimization problem to which we apply memcomputing to scale to the larger cases. Chapter 3 applies this technique to

accelerate an eventual supervised task, Chapter 4 applies it to the unsupervised case of restricted Boltzmann machines, and finally in Chapter 5 we discuss the extension of mode-assisted training to Deep Boltzmann machines, finding that mode assisted training performs as well or better than pre-trained or centered DBMs on the MNIST data set, all the while utilizing two orders of magnitude less parameters.

The dynamical systems induced by digital memcomputing machine designs explore phase spaces of complex optimization problems in a novel and highly advantageous way. This dissertation merely scratches the surface of the possible applications of these techniques. We conclude by looking toward the future, where there is much potential cross talk between memcomputing dynamics, artificial intelligence models, topological field theory, and neuroscience. Our journey begins with a historical detour to introduce the current Turing machine-based von Neumann architecture and discuss its conceptual and practical limits.

1.1 From Crisis to Computing: A Historical Prelude

At the turn of the 20th century, mathematics found itself amidst an existential crisis. Mathematicians found themselves asking such dangerous questions as 'What is mathematics, really?' [Fre93, RW10] and the slightly tamer, 'Is mathematics free of contradiction?' [Hil02] One of the leading approaches to these questions was from a philosophical school called *formalism*, which maintained that the whole of mathematical reasoning was essentially a formal game of symbols and consistent manipulation rules. Research in this direction culminated in what was broadly called *Hilbert's Program*: a systematic attempt, led by the influential mathematician and staunch formalist, David Hilbert, to distill mathematics into formal systems of rules, and prove their consistency. Doubt began to stir in the hearts of mathematicians when the first attempts to produce foundational theories of mathematics (i.e. set theory, logic) resulted in paradox[1].

[1] Perhaps the most famous example is Russell's Paradox in naive set theory: Does the set, $R = \{x | x \notin x\}$, of all sets, x, that do not contain themselves, contain itself? It does only if it does not, $R \in R \iff R \notin R$. Many others

Based on a conviction that there was no *ignoramibus* (unknowable) in mathematics, Hilbert was confident that with careful analysis these issues could be resolved. In 1928, Hilbert and his colleague Wilhelm Ackermann formulated yet another challenge: the *Entscheidungsproblem* (decision problem) [HA28]. They asked, does there exist an algorithm to determine whether or not a given mathematical statement is provable from a set of axioms?

The decade that followed was at least a series of checks on Hilbert's optimism if not a *coup de grâce* to his formalist program. In 1931, the German-Austrian logician Kurt Gödel, using recursive functions, proved the now famous incompleteness theorems[2] [Göd31], demonstrating the fundamental limits of provability in sufficiently expressive formal systems. Using λ-calculus, American logician Alonzo Church proved the Entscheidungsproblem was unsolvable in 1936 [Chu36]. Soon after, the young Alan Turing independently discovered the same truth, with his development of the Turing machine [Tur36]. All three methods were later found to render the same class of functions computable[3], leading to the Church-Turing hypothesis that *any* computable function in the informal sense is computable by a Turing machine. These results were an act of simultaneous destruction (in that they shattered the hopes of an absolute foundation encapsulating all truths in mathematics) and creation (they demarcated what was unknowable by establishing abstract notions of machines that can construct the knowable.) And thus like a phoenix, the modern idea of digital computation emerged from the rubble of this crisis.

---

contain similar flavors of self-referential mischief.

[2]The theorems demonstrated formal systems capable of expressing arithmetic over the natural numbers cannot: *i*) prove all true statements expressible by the system and *ii*) cannot prove their own consistency. To give a sense of the immediate impact of Gödel's result, it is said that physicist, computer scientist, polymath and fellow logician John von Neumann was in attendance when first Gödel publicly presented his proof. Upon its conclusion, von Neumann is reported to have exclaimed, "It's all over."

[3]Referred to now as *Turing-equivalence*.

1.2 Turing Machines and the von Neumann Architecture

Of the abstract frameworks, the Turing machine is closest to a physical device, and intentionally so. Turing wanted not only an abstract mathematical framework, but ultimately a *machine* to realize the algorithm called for in the Entscheidungsproblem. Turing's aim was to make mechanical the activity of a mathematician (or human computer) in solving a decision problem. The machine consisted of *i*) a *tape* separated into cells containing *symbols* from some alphabet. This represented the paper and symbols used by humans doing mathematics. On the tape was *ii*) a head that can read and write symbols to the tape, presumably the eyes and pen of the mathematician. A *iii*) state register stored the *state* of the machine, akin to the "state of mind" of the mathematician. And finally, a *iv*) finite table of instructions, or *transition function*, that determines if the head writes or erases a symbol, changes its state, or moves left or right, depending on the current symbol and internal state of the machine. In the same paper, Turing demonstrated the remarkable existence of *universal* Turing machines, which could simulate any other Turing machine by specifying a particular Turing machine as data on its tape. This made concrete the notion of the general purpose programmable computer.

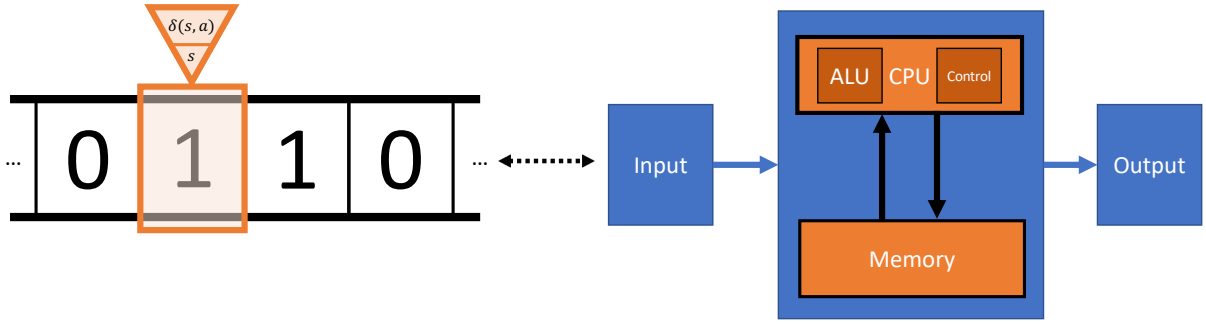


Figure 1.1: Anatomy of a Turing Machine, with transition function $\delta(s,a)$ and state, s, alongside the von Neumann architecture it inspired.

Universal Turing machines directly inspired the architecture of the first electronic, programable general purpose computer, the ENIAC. Outlined by John von Neumann in 1945 [vN93], the architecture which now bears his name logically separates where data and programs are

stored (the memory) and where the computations are performed (CPU). The von Neumann architecture, realized with transistor based silicon, is the large scale modern computing orthodoxy. It affords plenty of flexibility and ease of programability to create complicated control logic, since a program can compute on its own code same as a piece of data. The successful physical implementation of this organizational principle of digital computation was nothing short of a technological revolution, ushering in the information age which transformed almost every aspect of human life.

Computing architectures have not changed in their essential features since the 1950s, but many parts of the engineering of such systems has improved, resulting in handheld smartphones more capable than room sized supercomputers of the past. But this framework is now showing its age. The von Neumann architecture allows for flexible general purpose computation, but also introduces what is called the von Neumann bottleneck. Performance in these systems is limited by the constant need to communicate between CPU and memory[4], to obtain data as well as program instructions. This fundamental organizational limitation, as well as physical constraints brought on by Moore's 'law' scaling of transistors, and a growing set of unreachably difficult optimization problems, have together spurred the interest into unconventional computing architectures.

1.3 The Memcomputing Approach

In its broad strokes, memcomputing refers to computing with and in memory [DVT18], in contrast to conventional computing paradigms that physically and logically separate data, the instruction set (program), and where computations are actually performed (CPU). As an abstract computational model, both the universal (UMMs) and digital subclass of memcomputing machines (DMMs) were introduced as a collection of interconnected *memprocessors* [TD15, TDV17a], which store data and perform logical operations at the same physical location. Memcomputing

[4]Strictly speaking, this true of other computing architectures, like the Harvard architecture, as well.

architectures possess intrinsic parallelism, in that the transition function, δ, acts on the collective states of all memprocessors as well as computational overhead, which refers to the fact that the network topology encodes additional information used in the computation, and is not held by any individual memprocessor.

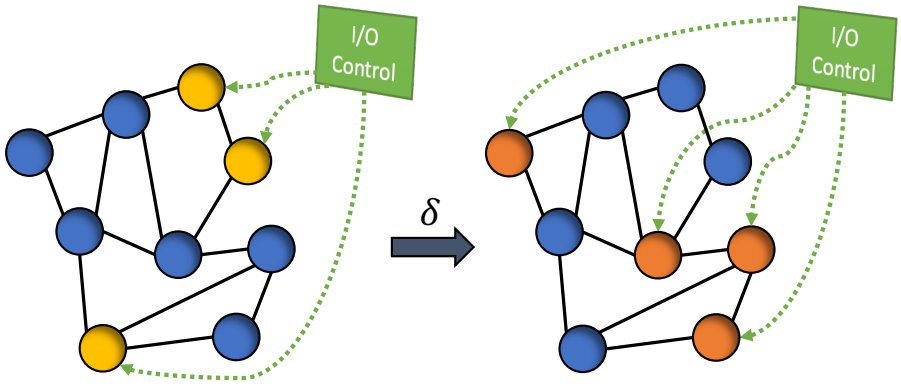


Figure 1.2: Operation of a digital memcomputing machine with m memprocessors. Input nodes of the network are initialized, then the transition function δ acts on the collective state of the network of memprocessors to produce output at the output nodes, read at the completion of the computation.

1.4 Self-Organizing Logic

In practice, DMMs can be realized through a dynamical generalization of Boolean circuits, called self-organizing circuits (SOLCs). These circuits are formed by replacing each traditional logic gate (e.g. AND, OR, NOT) in a Boolean circuit of interest by a dynamical equivalent self-organizing logic gate (SOLG). Each SOLG can be understood as a dynamical system, $\dot{\mathbf{s}} = \mathbf{F}(\mathbf{s})$, in an extended phase space, $\mathbf{s} = (\mathbf{v}, \mathbf{x})$, that includes the original voltages, $\mathbf{v}$, at the logic terminals as well as internal memory variables, $\mathbf{x}$. Whereas the original Boolean variables take on one of two discrete values, $v_i \in \{0, 1\}$, variables in SOLCs are relaxed to continuous values within some compact set. The usual choice is $x_i \in [0, 1]$ and $v_i \in [-1, 1]$, where $v_i \to 1$ corresponds to the logical 1 and $v_i \to -1$ to the logical 0. The flow field, $\mathbf{F}$ is designed such that the logically valid configurations of the gates correspond to the only attractive (stable) fixed points of the dynamics,

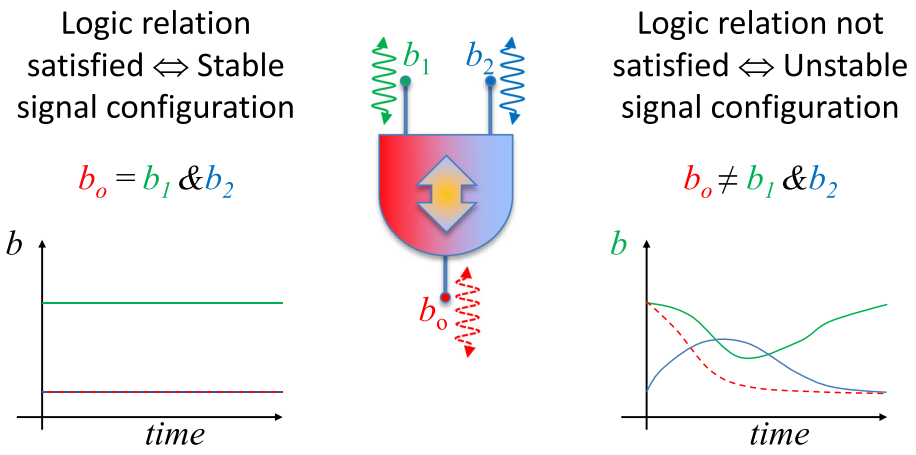


Figure 1.3: Simplified dynamics of a self-organizing logic gate. The gate is stable if the voltages at the gate terminals respect the appropriate logical relationship, otherwise the dynamics are unstable. Reproduced from [DVT18].

$\mathbf{F}(\mathbf{v},\mathbf{x}) = 0$. A simplified picture of the dynamics is shown in Figure 1.3. The particular choice of $\mathbf{F}$ is not unique, but must have certain properties, namely be *point dissipative*, to guarantee that the trajectory of an arbitrary initial condition will converge to the global attractor containing the prescribed fixed points [TDV17a].

Self-organizing logic gates and their resulting circuits are *terminal agnostic* in that any set of terminals can be regarded as input and/or output, and serve as a boundary conditions for the computation. This allows SOLCs to be run in both the typical forward mode, which evaluates the Boolean circuit, or in *inverse* mode, which solves for a logically valid input to the circuit corresponding to a desired output. It is this feature that makes SOLCs a fruitful realization of the memcomputing concept, as their native operation maximizes the number of satisfied gates in the circuit, which is known in the computer science community to be a highly expressive optimization problem called the *maximum satisfiability problem* or MAX-SAT. This problem belongs in the NP-hard complexity class of problems, which include Boolean satisfiability (SAT), the Traveling Salesman problem, finding the ground state of spin-glasses, and other combinatorial optimization problems of practical and theoretical interest[5].

[5]Computational complexity organizes problems into complexity classes depending on the resources a Turing machine requires to solve them. Problems considered informally 'tractable' by a Turing machine if they belong to

We give a simplified overview of the explicit construction of a SOLC in Ref. [TDV17a]. In Chapter 2, we apply these circuits to the problem of numerical inversion of floating point numbers.

1.5 Hardware Models

Initial models for self-organizing gates (SOLGs) that comprise SOLCs were designed directly as electric circuits, utilizing resistors, capacitors, voltage controlled voltage generators and memristors to incorporate memory. Memristors, or resistors with memory, are non-linear circuit elements whose resistance (or conductance) depends on the past history of current (or voltage) through the element. Their I-V relation can be written as the coupled dynamical system, also known as a *memristive system* [CK76],

$$I = g(x)V, \tag{1.1}$$

$$\dot{x} = f(x,V). \tag{1.2}$$

These components are organized to form dynamic correction modules (DCMs) at each logical terminal which act to adaptively enforce the desired constraint of the gate depending on the local voltage values at each terminal (spacial locality), as well as the past history of the terminal (temporal non-locality).

Each DCM contains voltage controlled voltage generators (VCVGs) whos voltage is determined by a linear combination of the voltages at each of the other terminals, with a DC offset,

$$v_{\mathrm{VCVG}} = a_0 v_0 + a_1 v_1 + a_2 v_2 + dc. \tag{1.3}$$

the class P, which is defined as all problems a Turing machine can solve in time that scales, to leading order, as a some polynomial in the size of the input, $O(n^k)$. All problems that a Turing machine can *verify* the solution of in polynomial time form the class NP. A problem X is considered C-hard for a complexity class C if any problem in C can be polynomial-time reduced to an instance of X, and is considered C-complete if $X \in$ C.

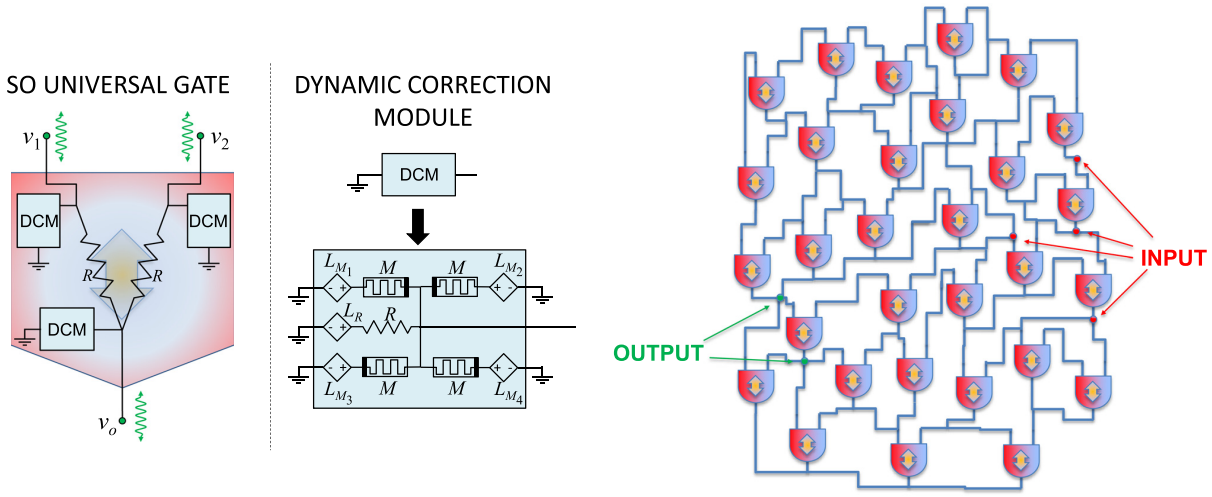


Figure 1.4: A three terminal self-organizing logic gate (left), a network of which forms a self-organizing logic circuit (right). The particular choice of gates and boundary conditions specify the problem to be solved. Reproduced from [TDV17a].

The coefficients are chosen such that when the gate is satisfied, no more current flows to any terminal, otherwise the DCM will inject current opposite the sign of the voltage at the terminal. Depending on the type of gate, up to four VCVGs in series with memristors as well as a resistor-VCVG pair determine the DCMs total current. The memristors are designed to increase the influence of a particular generator by increasing its conductance (or lowering their resistance) while a terminal remains unsatisfied. The resistor and VCVG pair pulls current when the gate is satisfied which drives the gate to equilibrium. A simplified version of the dynamics of a DCM can be written as the combined effect of the memristor-generator and resistor-generators in parallel,

$$C\dot{v} = g_m(x)v_m + g_r v_r \tag{1.4}$$

$$\dot{x} = f(x, v_m). \tag{1.5}$$

Where $g_m(x)$ is the conductance of the memristor as a function of its internal variable, x. A reasonable choice is $g_r(x) = (R_{on}(1-x)+R_{off}x)^{-1}$ which models a particular class of memristors. The dynamics of the internal variable, $f(x, v_m)$, is chosen such that the current from a DCM increases in strength when its associated terminal has been unsatisfied in the past. A numerically

friendly choice for current-driven memristors is

$$f(x, v_m) = -\beta(\theta(x)\theta(v_m) + \theta(1-x)\theta(-v_m))g(x)v_m. \tag{1.6}$$

Here $\theta(x)$ is the Heaviside step function, and β is a timescale typically set to be much slower than that that voltages. It is not obvious that the dynamics of a large SOLC avoids typical pitfalls of analog computation of this type, namely the proliferation of chaos or periodic orbits. It was shown that SOLCs are free of these phenomena, in the presence of solutions [TDV17a, DT17, DVT17].

The addition of slower timescale memory variables is essential, as they allow the system to adaptively raise or lower conductance at its VCVGs as needed to enforce the logical relationship between the nodes. The internal degrees of freedom given by the memory variables, in tandem with the continuous response of the voltages, allow the system to explore paths through phase space not accessible by other methods, say through illogical configurations. The dynamics can be generalized and simplified to other optimization problems, say finding ground states of spin-glasses [STDV19], which don't bare direct resemblance to Boolean circuits, although they can be mapped to one if needed. It was at first believed that SOLCs must be constructed in hardware to realize any computational advantage, but a surprising fact was discovered that in fact the numerical simulation of these systems are competitive with native digital algorithms for complicated NP-hard problems [SCTDV19, TCSD18, BPDV20]. In the next section, we discuss what properties of the dynamics give rise to this fact, but before that we provide a short comparison between the memcomputing concept and two other unconventional approaches to computing: quantum and neuromorphic computing.

Comparison to Quantum Computing

Memcomputing is, in a way, conceptually similar to the physics-based, Turing-alternative architecture, quantum computing [Ben80, Fey82]. Both approaches attempt to build systems

which inherit advantageous computational properties through judicious engineering of the underlying physical properties of a system. Quantum algorithms use superposition and entanglement inherent in quantum dynamics to accomplish a task (hopefully) more efficiently than a classical computer [Sho94]. In the case of memcomputing, it is the use of the emergent collective behavior of the individual voltages mediated by nonlinearity and memory dynamics. Both approaches are in a sense, analog computers, in that both computations utilize continuous dynamics. In between input and output, quantum dynamics follow the Schrödinger equation,

$$i\hbar\frac{\partial\Psi}{\partial t} = \hat{H}\Psi \tag{1.7}$$

Both approaches can be written to, and read digitally, and for all practical purposes, can be thought of as digital computers. However, in the case of quantum computing, the computation must be done while maintaining quantum coherence, which requires cryogenic temperatures and results in many engineering challenges. DMMs on the other hand can be built and ran at room temperature with classical circuit elements. Furthermore, DMMs have deep connections to quantum behavior which we briefly discuss in this next section. Through a topological field theory interpretation of their dynamics, their advantageous computational behavior can be seen as a classical form of quantum tunneling [DVTO17].

Comparison to Neuromorphic Computing

Neuromorphic computing and memcomputing both share inspiration from the biological brain, but have somewhat different goals as computing methodologies. Both approaches emphasize the fact that the brain seems to be using a dynamical and highly parallel, asynchronous method to compute – in stark contrast to the Turing-based von Neumann architecture.[6] Neuromorphic computing takes the perspective of attempting to emulate the basic functioning of

[6]To be fair to the progenitors Turing and von Neumann, this fact was not lost on them [TUR50, vN58].

the brain by reverse engineering its signaling mechanisms, a la spikes [WCB19], with the goals of recovering energy efficiency, building human-computer interfaces, and developing a greater understanding about biological brains as a result. In contrast, memcomputing does not attempt to recreate the brain in detail but rather to capture the relevant higher level dynamical properties that give rise to its efficient computational properties.

1.6 Memory Dynamics and Collective Behavior

The slower memory dynamics are timescale separated from the much faster voltages, effectively keeping the voltages in their local ground state[7] as the system evolves. From the perspective of the voltages, the memory dynamics are quasi-adiabatic, although the entire simulation heavily relies on the memory variables significantly evolving and adapting to the voltage dynamics. As the memory variables adjust, the voltages organize into intermittent periods of quiescence separated by rapid collective jumps, or avalanches, of voltages from one saddle point in the phase space to another. These avalanches possess a critical Borel distribution of cluster sizes, capable of flipping an extensive amount of variables in a way that lowers the energy [STDV19, BSDV19].

In dynamical systems language, they are families of heteroclinic trajectories between critical points of the flow field with different index.[8] These collective flips can be understood in a field theory context as instantons [DVTO17], which are families of trajectories that render the Euclidean action of the system stationary, $\delta S_E = 0$, and are the classical analog to quantum tunneling phenomena. The Euclidean action is related to the quantum tunneling amplitude by a Wick rotation of the propagator from one vacuum (local minima), $|x_i\rangle$, to another, $|x_f\rangle$,

[7] Or vacua in field theory language.

[8] A critical point of $\mathbf{F}$ is a fixed point, $\mathbf{F}(\mathbf{x}^c) = 0$, of the dynamics, $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$. $\mathbf{x}^c$ is stable (attractive, local minima) if the Hessian matrix is positive definite at $\mathbf{x}^c$, with eigenvalues $\mathrm{Re}(\lambda_i) > 0$, versus a saddle point where the Hessian is indefinite, with positive (stable) and negative (unstable) eigenvalues $\mathrm{Re}(\lambda_i) \in (-\infty, \infty)$. The *index* of a critical point is the number of unstable directions at $\mathbf{x}^c$, or eigenvalues with negative real part.

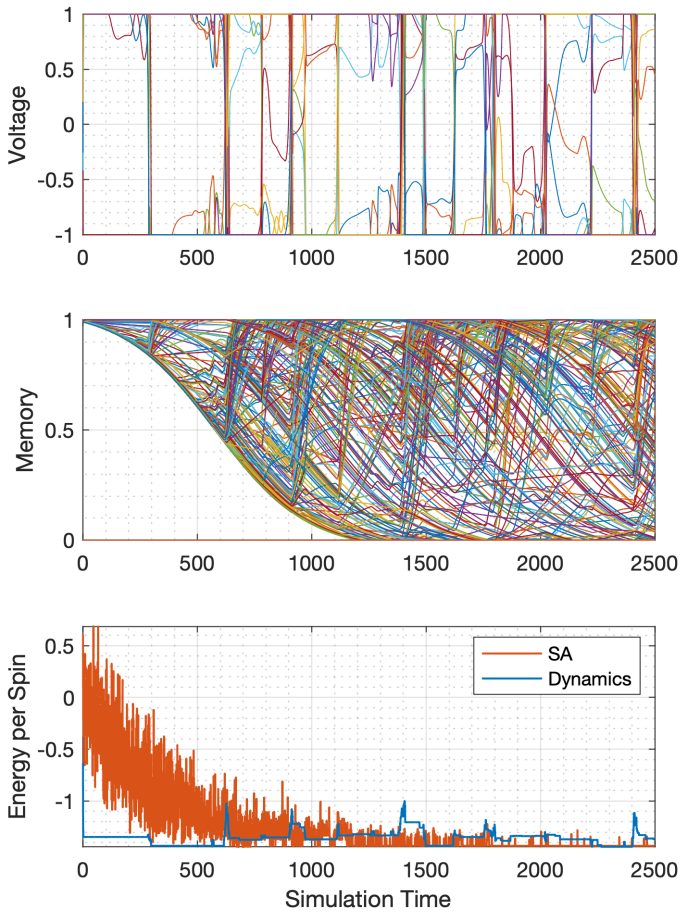


Figure 1.5: DMM optimization performance on a 25 variable spin-glass Hamiltonian (Eq. 1.10) problem compared to simulated annealing. Couplings and external field are sampled uniformly as $J_{ij}, b_i \sim U[-1,1]$

$$\langle x_f | e^{-i\hat{H}t/\hbar} | x_i \rangle = \int_{x(0)=x_i}^{x(T)=x_f} \mathcal{D}[x(t)] e^{\frac{i}{\hbar} S[x(t)]} \tag{1.8}$$

$$\overset{t \to -it}{=} \int \mathcal{D}[x(t)] e^{-\frac{1}{\hbar} S_E[x(t)]} \tag{1.9}$$

The introduction of memory variables in the circuit has the effect of turning every local minima of the circuit into a saddle point and the coupled dynamics between voltage and memory

allows an active augmentation of the curvature around a critical point, allowing the system to fall into the basin of attraction of yet another saddle point, and quickly relax to it (instantonic jump). Crucially, this process couples more and more spins as time evolves, allowing jumps that flip an extensive number of spins toward lower energies, a classical form of tunneling. This makes accessible regions of phase space that would be impractical to reach with local methods [STDV19]. This phenomenon was shown to occur at the level of the self-organizing logic gates [BMTD18].

In Figure 1.5, the numerical simulations of spin-glass optimization by DMM dynamics is presented, including voltages, memory variables and the associated energy per spin as a function of simulation time. The results are compared to a popular stochastic optimization algorithm, simulated annealing. In this small dimensional problem, both methods find the ground state of the spin-glass, but in order to do so, both methods need to escape local minima in this non-convex problem. Simulated annealing uses random moves to move up in energy and escape local minima of the non-convex problem. On the other hand, DMM dynamics are completely deterministic and utilize auxiliary memory variables to 'tunnel' through energy barriers. Finally, with simulated annealing, the jumps are random, so as the system grows this method will lose its performance, whereas memory variables collectively move an extensive number of spins in the DMM, it can scale much more favorably to larger systems.

In the next section, we describe how these collective dynamics (spacial non-locality) mediated by memory (time non-locality) greatly aid in the training of the influential artificial intelligence model, the Boltzmann machine.

1.7 Of Neural Networks and Boltzmann Machines

From improvements in computer vision [LB95] and natural language processing [BMR$^+$20], to surpassing human performance in the game of Go [SSS$^+$17], it is hard to overstate how much

attention is drawn by the latest achievements in artificial intelligence. This most recent wave of new results, called the 'deep learning revolution,' has been built on the backs of networks of simplified models of neurons [MP43], called artificial neural networks (ANNs) [LBH15]. Coupled with a data set, ANNs can be 'trained' by tuning their internal parameters to iteratively become better at some objective defined by a loss function.

One of the most highly influential models in this class is the Boltzmann machine [HSA84, Smo86]. It is said that the discovery of effective means to train these models to state-of-the-art performance that ushered in the deep learning era of AI [HOT06]. They can be used as a compact generative model for complex probability distributions, a critical task throughout in many unsupervised learning settings. Variants of the Boltzmann machine have found applications in recommender systems [SMH07], and can even serve as powerful variational representations of many-body quantum states [CT17]. Boltzmann machines are a stochastic generalization of Hopfield networks [Lit74, Hop82], which themselves are recurrent[9] neural networks directly inspired by Ising-like models of ferromagnetism. Both models can be understood as a set of pairwise interacting classical spins with interaction strength, J_{ij}, in the presence of an external field, b_i, with Hamiltonian,

$$H(\mathbf{s}) = -\frac{1}{2}\sum_{ij} J_{ij} s_i s_j - \sum_i b_i s_i. \tag{1.10}$$

It was discovered that these systems are a fruitful model of associative memory, and could 'store' persistent states by tuning interaction energies, J_{ij}. Weights can be specified by a Hebbian learning rule[10], $J_{ij} = \frac{1}{N}\sum_n (2s_i^n - 1)(2s_j^n - 1)$, where the sum is over all $n = 1, \ldots N$ desired states.

[9] A recurrent neural network is one where the corresponding graph contains cycles or feedback loops, as opposed to feedforward neural networks which are directed acyclic graphs.

[10] Hebbian learning, named after Donald Hebb, is a theory of synaptic plasticity, or how neurons adapt their connectivity in learning environments. Frequently summerized as "Neurons that fire together, wire together."

Spins follow dynamics of simplified artificial neurons [MP43],

$$s_i = \begin{cases} +1, & \sum_j J_{ij} s_j + b_i > 0 \\ -1, & \text{otherwise.} \end{cases} \tag{1.11}$$

When the interactions are symmetric, $J_{ij} = J_{ji}$, the energy function given in Eq. 1.10 is a Lyapunov function for the dynamics. From an initial configuration, repeated updates evolve the system to lower and lower energy states, approaching the closest attracting fixed point given by a local minimum of $H(\mathbf{s})$, which are dominated by the chosen configurations. However, the dynamics can get 'stuck' in local minima that do not correspond to any desired configuration, called spurious minima, which inspired the development of Boltzmann machines.

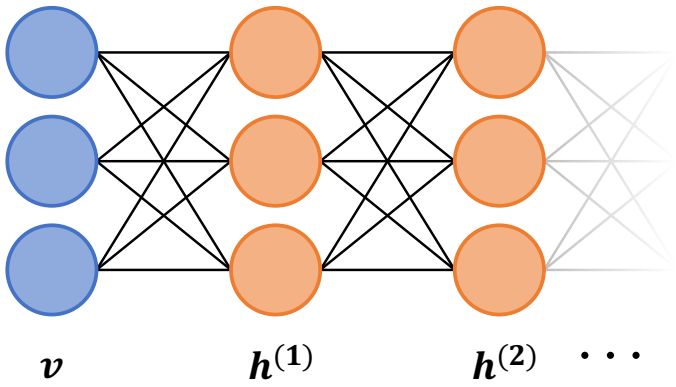


Figure 1.6: Graphical representation of a deep Boltzmann machine with one visible layer (blue) and two hidden layers (orange). Connections between nodes are undirected and symmetric, unlike typical feedforward neural networks. Reproduced from Ref. [MDV21]

Boltzmann machines differ from Hopfield networks in two important ways. First the artificial neuron dynamics are now stochastic, which allows the network to escape from undesired local minima. And secondly, in addition to *visible* nodes fixed by the data, Boltzmann machines introduce *hidden* nodes, which are not specified by the data. This adds the ability to model higher order correlations among the visible layers rather than only pairwise ones. The Hamiltonian, or energy function, of a BM is identical to that of Eq. 1.10, which written in terms of visible and

hidden nodes becomes,

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i>j} V_{ij} v_i v_j - \sum_{i>j} W_{ij} v_i h_j - \sum_{i>j} H_{ij} h_i h_j - \sum_i a_i v_i - \sum_i b_i h_i. \tag{1.12}$$

For computational efficiency, various subclasses of BMs have been introduced. The most popular being the restricted Boltzmann machine (RBM) [Hin02], specified by removing connections within a layer, $V_{ij} = H_{ij} = 0$, and the deep Boltzmann machine (DBM) [SH09], produced by setting $V_{ij} = 0$ and organizing the hidden nodes into layers.

Since BMs operate at a finite temperature, this induces a distribution over states given by the Boltzmann-Gibbs distribution,

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\mathcal{Z}} \tag{1.13}$$

This is familiar in statistical mechanics as the canonical ensemble, where the normalization constant, $\mathcal{Z} = \sum_{\{\mathbf{v},\mathbf{h}\}} e^{-E(\mathbf{v},\mathbf{h})}$, is known as the partition function. It is the distribution of states of a system with Hamiltonian Eq. 1.12 in contact with a heat bath, in this case with inverse temperature parameter, $\beta = 1$. The conditional distribution of each node is dependent only on the activity of its neighbors, and is given by a sigmoid of the sum of all inputs,

$$p(v_i = 1|\mathbf{h}) = \sigma\left(\sum_j W_{ij} h_j + a_i\right) = \frac{1}{1 + e^{-\sum_j W_{ij} h_j - a_i}} \tag{1.14}$$

The marginal distribution over the visible layer is obtained my tracing out the values of the hidden nodes,

$$p(\mathbf{v}) = \frac{1}{\mathcal{Z}} \sum_{\{\mathbf{h}\}} e^{-E(\mathbf{v},\mathbf{h})} \tag{1.15}$$

In the learning setting, the Boltzmann machine is tasked to match this model distribution to that of a data set, $\mathcal{D}$, drawn from some unknown distribution $q(\mathbf{v})$. This is typically framed by minimizing the Kullback-Leibler divergence between the unknown data distribution, q, and the

BMs model distribution, p,

$$\mathrm{KL}(q||p) = \sum_{\{\mathbf{v}\}} \log q(\mathbf{v}) \log \frac{q(\mathbf{v})}{p(\mathbf{v})} \tag{1.16}$$

Or equivalently, maximizing the *log-likelihood* of the data set, $\mathcal{D}$,

$$\begin{aligned} \mathcal{L} &= \sum_{\mathbf{v}\in\mathcal{D}} \log p(\mathbf{v}) \\ &= H(q) - KL(q||p) \end{aligned} \tag{1.17}$$

Where $H(q)$ is the Shannon entropy of q. Both of these expressions are highly non-linear functions of the BM parameters, and usually high dimensional. As such, the only practical approaches are found to be variants of stochastic gradient optimization, which gives a tidy learning rule as the difference between two expectations,

$$\Delta W_{ij} \propto \frac{\partial \mathcal{L}}{\partial W_{ij}} = \nabla \log \tilde{p}(\mathbf{v}) - \nabla \log \mathcal{Z} \tag{1.18}$$

$$= \langle v_i h_j \rangle_{\mathrm{D}} - \langle v_i h_j \rangle_{\mathrm{M}} \tag{1.19}$$

Each synaptic weight depends only on the local average activity of the two neurons it connects, under two different distributions. This kind of learning rule is called 'local,' a mostly desired property and biologically plausible, since the derivative of a weight is only dependent on the nodes it connects, using no 'non-local' information The first expectation is called the *data* term, and is under the distribution $q(\mathbf{v})p(\mathbf{h}|\mathbf{v})$, with the visible nodes fixed by the data. This is a Hebbian term which pushes up on W_{ij} if the two neurons it connects are 'on' often together. This term has the effect of *lowering* the energy (makes more likely) of configurations that match the data set on the visible layer. The second term is called the *model* term and is taken with respect to the full Boltzmann distribution, $p(\mathbf{v},\mathbf{h})$ with all nodes free to vary. This is a anti-Hebbian term that *raises*

the energy (makes less likely) of configurations that the model ascribes high probability. Learning completes when these two forces balance, and the BM has learned the distribution of the data.

Estimating the data term is tractable in some models (RBM), but the model term (responsible for minimizing the partition function) is usually intractable for all models. Sampling difficulties can be partially understood by the fact that a BM with weight sampled from a Gaussian distribution is equivalent to the Sherrington-Kirkpatrick spin glass. [SK75] This model was shown to possess an unwieldy free energy landscape [MPV87]. At low temperatures, during the later stages of BM training when weights become large, the free energy contains exponentially many local minima satisfying ultrametricity, separated by large energy barriers which break ergodicity, and self-averaging. This all but guarantees exponentially long equilibration times for any Markov Chain Monte Carlo approach.

A particular implementation of Gibbs sampling on RBMs, called contrastive divergence (CD-k), initializes short chains of length k with the data set. Usually $k = 1$ is found to be good enough of a signal for training. In even modestly sized networks, however, CD possesses fatal drawbacks. Since it initializes chains from the data set, density that accumulates far from the data, referred to as spurious modes, and can cause divergence in training. The random walk exploration of CD, and the combinatorial explosion of the phase space of BMs, make for incredibly long mixing times, which any Gibbs-like algorithm will be susceptible to.

1.8 Mode-Assisted Training

Mode-Assisted training probabilistically mixes samples of the mode of the BM with regular CD updates to achieve better performance and capture more model capacity than Gibbs sampling alone. Specifically built to correct the spurious mode problem with CD, it allows CD to sample excited states effectively, while finding and correcting any density accumulated away from the data set by a specialized optimization technique – memcomputing in this case.

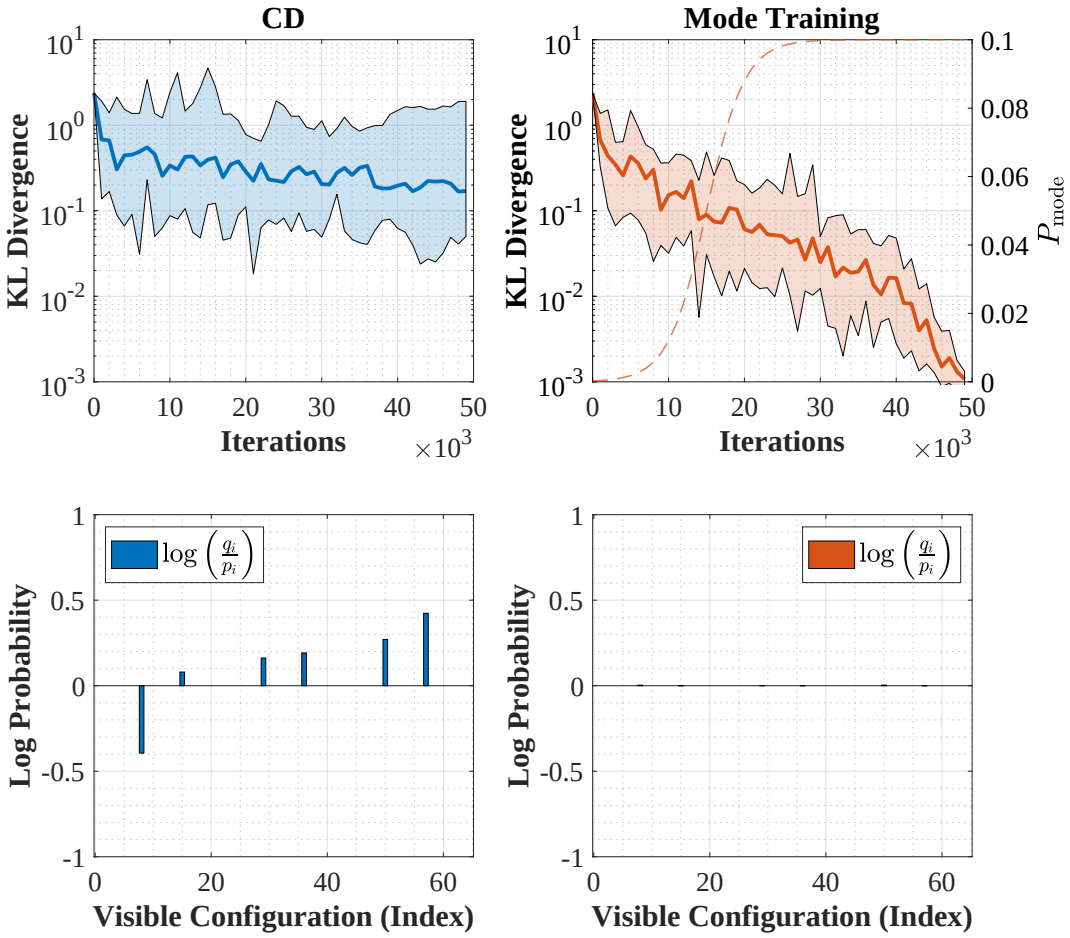


Figure 1.7: KL divergence comparison between mode-assistance and CD on an ensemble of twelve 6×6 RBMs trained on the shifting bar data set. The solid line in the first row denotes the median KL achieved at that point in training, and the max/min define the shaded area.

Mode-assistance consists of probabilistically replacing the standard CD approximated expectations in the log-likelihood gradient with samples of the mode of the respective distributions instead,

$$\Delta W_{ij} \propto \begin{cases} \langle x_i x_j \rangle_{\mathrm{D}} - \langle x_i x_j \rangle_{\mathrm{M}} \\ [x_i x_j]_{\mathrm{D}} - [x_i x_j]_{\mathrm{M}} \end{cases} \tag{1.20}$$

Incorporating mode samples in a probabilistic way with CD, with more samples in the later stages of training, is found to perform particularly well. The probability of a mode sample,

as a function of gradient iteration is taken to be a sigmoid,

$$P_{\text{mode}}(n) = P_{\text{max}}\sigma(\alpha n + \beta). \tag{1.21}$$

The parameters α and β determine the shape of the sigmoid and P_{max} determines the maximum probability that a mode sample will be used in training. The basic idea behind mode-assisted training is to address the main failure point of CD or MCMC methods in multimodal, highly dimensional spaces. When weights grow larger during training, and the model 'cools' to a lower temperature, the Gibbs chains get trapped very easily, and furthermore, the expectation becomes more and more dominated by lower energy states, while higher ones remain exponentially suppressed.

A mode sample can be thought of as a saddle point approximation to the expectation,

$$\langle f(\mathbf{x})\rangle_p = \frac{\int e^{-E(\mathbf{x})} f(\mathbf{x}) d\mathbf{x}}{\int e^{-E(\mathbf{x})} d\mathbf{x}} \approx f(\mathbf{x}_{\text{mode}}) \tag{1.22}$$

Later on in training, this approximation becomes dominant, and is why mode-assistance is most effective later in training when CD performs particularly poorly.

A comparison of training progress between CD and mode-assisted training on a 6×6 RBM is shown in Fig. 1.7. Even in small model sizes, CD can struggle with learning the correct probability distribution. Mode-assistance on the other hand, converged on the correct distribution with two orders of magnitude smaller KL divergence compared with CD. This benefit of mode-assistance is exaggerated in larger models, where CD not only misses the correct amplitudes across the support of the data set, but suffers from density accumulating away from data (spurious modes).

Finding the mode of an RBM or DBM is itself an NP-hard optimization problem. Although other optimizers can in principle be used (like simulated annealing), a particular DMM developed for related problems performs favorably compared to these approaches, and is what we

use to sample the mode of larger BMs [MPBDV20]. Even when accounting for the numerical overhead of integrating the memcomputing dynamics to find the mode, mode-assisted training does better that CD-k alone. From this we take that in large and complex training scenarios, it is better to use a specialized optimizer to find and correct areas of accumulated density (modes) rather than wait for a Markov chain to equilibrate.

The mode-assisted advantaged is realized in several deep learning contexts. In chapter 3, we find that stacks of mode-assisted RBMs (called a Deep Belief Network) get an advantageous starting point for back-propagation in supervised learning tasks [MTDV19], compared to random initial conditions. Chapter 4 finds that mode-assisted RBMs also outperform PCD, PT and other RBM training frameworks and achieve better log-likelihoods on synthetic data sets and the MNIST data set [MPBDV20]. Finally, in chapter 5 we find that mode-assisted training leads to dramatic improvements to DBM training compared to pre-training and centered DBMs, and achieves the same or better performance on the MNIST data set all while utilizing two orders of magnitude fewer parameters [MDV21].

Chapter 2

Memcomputing Numerical Inversion with Self-Organizing Logic Gates

Abstract

We propose to use Digital Memcomputing Machines (DMMs), implemented with self-organizing logic gates (SOLGs), to solve the problem of numerical inversion. Starting from fixed-point scalar inversion we describe the generalization to solving linear systems and matrix inversion. This method, when realized in hardware, will output the result in only one computational step. As an example, we perform simulations of the scalar case using a 5-bit logic circuit made of SOLGs, and show that the circuit successfully performs the inversion. Our method can be extended efficiently to any level of precision, since we prove that producing n-bit precision in the output requires extending the circuit by at most n bits. This type of numerical inversion can be implemented

by DMM units in hardware, it is scalable, and thus of great benefit to any real-time computing application.

2.1 Introduction

In recent decades, a growing interest into novel approaches to computing has been brewing, leading to several suggestions such as quantum computing, liquid-state machines, neuromorphic computing, etc.[LJ09] [ILBH$^+$11] [NC10]. Along these lines, a new computational paradigm has been recently introduced by two of us (FLT and MD), based on the mathematical concept of Universal Memcomputing Machines (UMMs) [TD15]. This novel approach utilizes memory to both store and process information (hence the name "memcomputing" [DVP13a]), and in doing so it has been shown to solve complex problems efficiently [TRBDV15]. The fundamental difference with Turing-like machines as implemented with current architectures, i.e., von Neumann, is that the latter do not allow for an instruction fetch and an operation on data to occur at the same time, since these operations are performed at distinct physical locations that share the same bus. Memcomputing machines, on the other hand, can circumvent this problem by incorporating the program instructions not in some physically separate memory, but encoded within the physical *topology* of the machine.

UMMs can be defined and built as fully analog machines [TRBDV15]. But these are plagued by requiring increasing precision depending on the problem size for measuring input/output values. Therefore, they suffer from noise issues and hence have limited scalability. Alternatively, a subset of UMMs can be defined as *digital* machines [TDV17a]. Digital Memcomputing Machines (DMMs) map integers into integers, and therefore, like our present digital machines are robust against noise and easily scalable. A practical realization of such machines has been suggested in Ref. [TDV17a], where a new set of logic gates, named self-organizing logic gates (SOLGs), has been introduced. Such logic gates, being non-local in time, can *adapt* to

signals incoming from *any* terminal. Unlike the standard logic gates, the proposed SOLGs are non-sequential, namely they can satisfy their logic proposition irrespective of the directionality in which the signal originates from: input-output literals or output-input literals. When these gates are then assembled in a circuit with other SOLGs (and possibly other standard circuit elements) that represents – via its topology – a particular Boolean problem, they will *collectively* attempt to satisfy all logic gates at once (intrinsic parallelism). After a transient that rids of the "logical defects" in the circuit via an instantonic phase [DVTO17], and which scales at most polynomially with the input size, the system then converges to the solution of the represented Boolean problem [TDV17a].

In this paper, we suggest to apply these digital machines within their SOLG realization to numerical scalar inversion. We will also briefly mention how to generalize this approach to linear systems and matrix inversion. These problems scale polynomially with input size, so one would expect our present work to be of limited benefit. On the contrary, numerical scalar and matrix inversion, and solving linear systems are serious bottlenecks in a wide variety of science and engineering applications such as real-time computing, optimization, computer vision, communication, deep learning, and many others [Sze11] [Dat10] [Sch15] [TDH16] [WSG94]. An efficient hardware implementation of such numerical operations would then be of great value in the solution of these tasks.

The rules for basic numerical manipulations for engineering and scientific applications are given by the IEEE 754 floating point standard [IEE08]. In modern day processors, alongside the arithmetic logic units which operate on integers, there exist floating-point units (FPUs) that operate on floating point numbers with some given precision [Int16]. Whilst addition and multiplication operations are quite optimized in hardware, floating-point division, which typically employs a Newton-Raphson iteration[OF97], performs three to four times slower in latency and throughput.[1] In the case of matrices, inversion is typically performed in software (LAPACK, BLAS). Most

[1] Latency - Number of clock cycles it takes to complete the instruction. Throughput - Number of clock cycles required for issue ports to be ready to receive the same instruction again.

algorithms must continuously communicate between the CPU and memory, eventually running into the von Neumann bottleneck. This is a prime concern for software matrix inversion. In fact, methods to minimize communication with memory to solve linear systems is an active area of research. [BDHS11] In recent years, real-time matrix inversion has been implemented in hardware by FPGAs for wireless coding. [MDMM11] Even a quantum algorithm to speed up the solution of linear systems has been proposed [HHL09]. However, these hardware solutions are typically constrained to very small systems or cryogenic temperatures, respectively.

In this paper, we avoid many of these issues by introducing a fundamentally different approach to scalar (and matrix) inversion. We propose solving the inversions in hardware by employing DMMs implemented with self-organizable logic circuits (SOLCs), namely circuits made of a collection of SOLGs. In this way, the need for a complicated approach to inversion is greatly simplified since the 'algorithm' implemented is essentially a Boolean problem, and the computational time is effectively reduced to real time when done in hardware. Our novel, and non-trivial contribution here is extending the factorization solution in [TDV17a] to a circuit which inverts fixed point scalars. We demonstrate that this can be done efficiently at any desired precision by proving that requiring n bits of precision in the result demands extending the factorization circuit by at most n more bits. To the authors' knowledge, this is the first explicit construction of a fixed point inversion, with an arbitrary specified precision, into an exact factorization between integers. We also provide a roadmap from our scalar inversion method toward full matrix inversion and the solution of linear systems, the full implementation of which is left for future work.

This paper is organized as follows: In Sec. 2 we briefly outline the concept of DMMs and SOLCs. In Sec. 3 we describe in detail how the scalar inversion problem can be solved so that the reader can follow all its conceptual and practical steps without being bogged down by details. In this section we also simulate the resulting circuit for several cases, and discuss the scalability of our approach. In Sec. 4 we describe how matrix inversion can be done by repeated application of our solution for scalars. Finally, in Sec. 5 we report our conclusions.

2.2 Digital Memcomputing Machines and Self-Organizing Logic Circuits

The method we employ for numerical inversion is an extension of the factorization solution done by Traversa and Di Ventra in Ref. [TDV17a]. We build on their method to perform scalar inversion, and by generalization, find the inverse of a matrix.

The SOLCs we utilize here are a practical realization of DMMs, which themselves are a subclass of UMMs, and thus take advantage of *information overhead*, namely the information embedded in the topology of the network rather than in any memory unit, and the *intrinsic parallelism* of these machines, which refers to the fact that the transition function of the machine acts simultaneously on the collective state of the system [TD15]. Digital memcomputing machines can be formally defined in much the same way as Turing machines, as the following eight-tuple: [TDV17a]

$$\text{DMM} = (\mathbb{Z}_2, \Delta, \mathcal{P}, S, \Sigma, p_0, s_0, F). \tag{2.1}$$

Here Δ is a set of transition functions,

$$\delta_\alpha : \left(\mathbb{Z}_2^{m_\alpha} \setminus F\right) \times \mathcal{P} \to \left(\mathbb{Z}_2^{m'_\alpha} \setminus \mathcal{P}^2\right) \times S, \tag{2.2}$$

where S is a set of indices α, $\mathcal{P}$ is a set of arrays of pointers p_α that select memprocessors called by δ_α, Σ is a set of initial states, p_0 an initial array of pointers, s_0 the initial index, and F, a set of final states.

From the above abstract definition we take two concrete points relevant to our discussion here. The first point is the fact that δ_α, the transition functions, act on the *collective states* of the memprocessors, which gives rise to the intrinsic parallelism of the DMM. They also map finite sets of states to finite sets and hence describe a digital system. Secondly, we can explicitly see the novelty of DMMs, and how they differ from Turing machines. The way the DMM works is by first

encoding a given problem into the topology of the network of memprocessors, which specifies the structure of the set $\mathcal{P}$, which in turn facilitates a non-trivial communication between states by the transition functions. Thus, the DMM works by leveraging the structure of a given problem as reflected in the topology of the machine. This gives rise to the information overhead, and not in any additional software or instructions. This makes the DMM a special purpose machine designed to solve a specific problem in an efficient way. Compare this to a Turing machine, which is a more general computational paradigm but lacks the ability to specialize its processing to a given task, without some external instruction set.

All this is realized in practice by SOLCs [TDV17a]. In these circuits, the computation is performed by first constructing the "forward problem" with standard logic gates, namely the Boolean circuit that would solve such a problem, if the result were known. This specifies the topology of the DMM. To be more precise, if we let $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ be a system of Boolean functions where $\mathbb{Z}_2 = \{0, 1\}$, then we ask to find a solution $\mathbf{y} \in \mathbb{Z}_2^n$ of $f(\mathbf{y}) = \mathbf{b}$. If numerical inversion is such problem and we replace the gates of the Boolean functions with those that self-organize, the SOLC so constructed would find $f^{-1}(\mathbf{b})$, its solution.

There is a fundamental difference between standard networks of logic gates and ones that self-organize, in that the latter ones allow for any combination of input/output satisfiable in the Boolean sense, i.e., *non contradictory*. One can, in principle, provide these logic circuits with combinations that are not satisfiable. In this case, the system will not settle to any fixed (equilibrium) point. For the formal analysis of these dynamical systems and convergence properties, we refer the reader to the details in Ref. [TDV17a], and Ref. [DVT17] where it was demonstrated that chaotic behavior cannot emerge in DMMs with solutions. We stress that this is not the same as a "reversible logic gate" that requires the same number of input and output literals and an invertible transition function [SPMH03].

From a physical point of view, the entire network of gates acts as a continuous dynamical system and self-organizes into a global attractor, the existence of which was demonstrated in

[TDV17a]. This network of logic gates exploits the spatial non-locality of Kirchhoff's laws and the adaptability afforded by the time non-locality to find the solution of a given problem in one computational step if implemented in hardware. The fundamental point is that although the computation occurs in an analog (continuous) sense, SOLCs represent a scalable *digital* system since both the input and output are written and read digitally, namely require a *finite precision*.

2.3 Detailed Analysis of Scalar Inversion

In this section, we outline a detailed construction of a system of self-organizing logic gates which solves a scalar inversion problem consisting of fixed precision binary numbers. In terms of circuit topology, scalar inversion is almost bitwise the same as scalar multiplication, and thus our inversion circuit looks similar to the factorization circuit it is based on. However, in the inversion case a few more complications emerge. Namely, how does one map a general scalar inversion problem onto a logical circuit, and hence an exactly satisfiable problem?

This problem can be solved by constructing an "embedding" into a higher dimensional space where the arithmetic is between natural numbers, and hence exact. We do this by padding the problem with an extra register of *satisfiability* bits. Also, one must be especially careful with the interpretation of the input and output since the bitwise input into the circuit does not map transparently into the numerical values of input and output.

To clarify all this, let us consider the problem of inverting a scalar, say $a \times b = c$. We are given a and c with fixed-point exponents and n-bit mantissas where we normalize the leading bits of a and c to be 1, so we are guaranteed that there is always a solution. We forgo the discussion of the sign, as the sign bit of the product is simply an XOR between the sign bits of the constituents. The task before us is to perform an analysis of the forward problem, making sure that we translate the problem into a satisfiability (SAT) problem, since this is essentially what the SOLC solves.

In its most general form the problem is:

$$2^{m_a}(0.1a_{n-2}\cdots a_0)\times 2^{m_b}(0.b_{n-1}\cdots b_0)=2^{m_c}(0.1c_{n-2}\cdots c_0), \tag{2.3}$$

where a_i, b_i and c_i in the above equation are either 0 or 1.

We immediately see that the following relationship between the exponents holds: $m_a + m_b = m_c$. By setting the unknown exponent value to be $m_b = m_c - m_a$, we are left with only the relationship with the mantissas.

What remains is a detailed analysis of the scalar inversion of the binary mantissas which we now consider independently of the exponent, essentially treating the arithmetic between natural numbers – where each mantissa can be reinterpreted explicitly as an integer. For example, the n bits of a would be reinterpreted now as,

$$a = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_0 2^0 \tag{2.4}$$

The same procedure would be performed with b and c.

It is obvious that to satisfy the consistency of the arithmetic, the size of the mantissa of c needs to be equal to the sum of the number of bits of a and b. To that effect, we must add n bits – which we refer to as *consistency* bits – to c which we set to zero.

There now remain two issues. Since this is arithmetic between natural numbers, it must be exact, and under the current constraints it is not always the case (take $a = 3$ and $c = 1$, in digital representation, for example). To address this issue, we pad both b and c with what we call *satisfiability* or SAT bits, labeled b_f and c_f respectively, which are not specified and are allowed to float. This is to give the problem enough freedom to be exactly satisfiable in the Boolean sense (and hence solvable by a SOLC). Below we address the issue of how many such bits one needs to ensure exact satisfiability.

There is finally the issue of the accuracy of our answer b in bits. In order to control

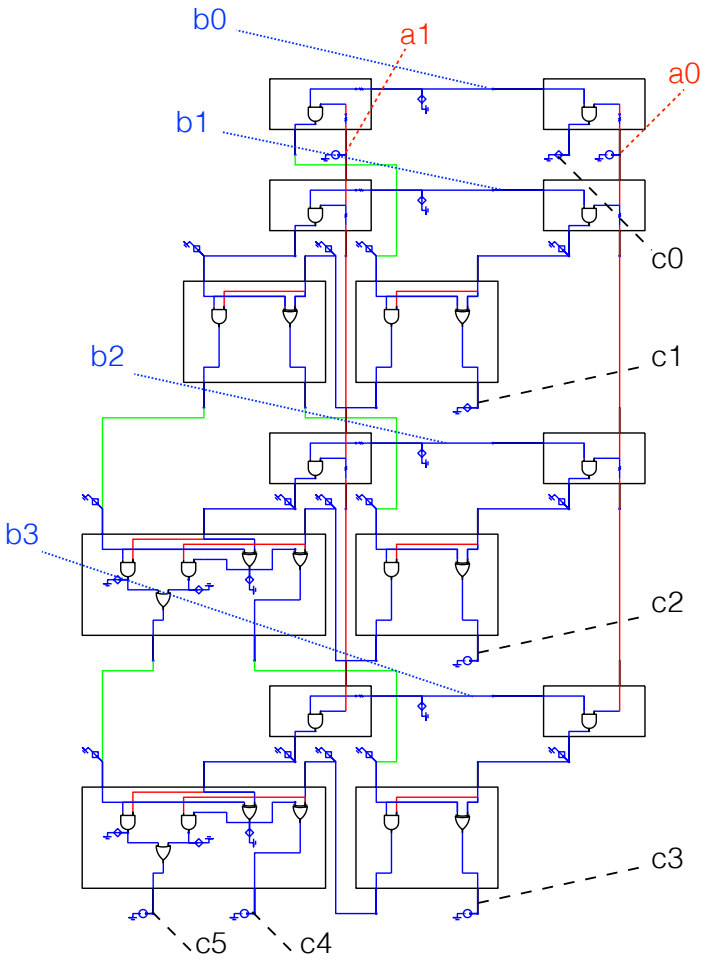


Figure 2.1: A schematic of a 2-bit inversion circuit. The nodes represent voltages interpreted as the bits of the input/output. The nodes are connected to a series of 2- and 3-bit adders which form the product $a \times b$ connected to the resulting bits of c.

precision, we pad a with an n_a number of zeros or *enhanced precision* bits, which we actually show do not change the solution.

We now show all these steps explicitly. In a more compact representation, and keeping in mind the above definitions, we write our original problem as

$$\underset{n}{[\mathbf{a}]}\underset{n_a}{[\mathbf{0}]} \times \underset{n}{[\mathbf{b}]}\underset{n_b}{[\mathbf{b_f}]} = \underset{n}{[\mathbf{c}]}\underset{n}{[\mathbf{0}]}\ \underset{n_a+n_b}{[\mathbf{c_f}]} \tag{2.5}$$

Here a, b and c represent the bits of the mantissas, b_f and c_f represent the floating bits, and n_a

and n_b represent the size of the accuracy register and floating bit register, respectively. We are essentially constructing an "embedding" of the problem to one in a higher dimensional space where the arithmetic can indeed be satisfied exactly. The goal is to then 'project' back, or truncate, onto the original space to obtain b with the desired accuracy.

The embedding is given by the following injective map where $\tilde{a}$ and $\tilde{c}$ represent the precision and consistency bits of zeroes,

$$\begin{aligned} \hat{a} &= a2^{n_a} + \tilde{a} \\ \hat{b} &= b2^{n_b} + b_f \\ \hat{c} &= c2^{n_a+n_b+n} + \tilde{c}2^{n_a+n_b} + c_f \end{aligned} \tag{2.6}$$

Since $\tilde{a}$ and $\tilde{c}$ are identically zero, the inversion we solve, $\hat{a} \times \hat{b} = \hat{c}$ is written,

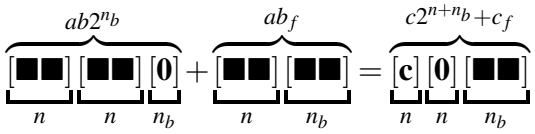
$$\begin{aligned} a2^{n_a}(b2^{n_b} + b_f) &= c2^{n_a+n_b+n} + c_f \\ ab2^{n_a+n_b} + ab_f 2^{n_a} &= c2^{n+n_a+n_b} + c_f \end{aligned} \tag{2.7}$$

We know that the number of bits of the left in Eq. (2.7) must be equal to the number of bits on the right. The problem then becomes:

$$\overbrace{\underbrace{[\blacksquare\blacksquare]}_{n}\underbrace{[\blacksquare\blacksquare]}_{n}\underbrace{[\mathbf{0}]}_{n_b}\underbrace{[\mathbf{0}]}_{n_a}}^{ab2^{n_a+n_b}} + \overbrace{\underbrace{[\blacksquare\blacksquare]}_{n}\underbrace{[\blacksquare\blacksquare]}_{n_b}\underbrace{[\mathbf{0}]}_{n_a}}^{ab_f 2^{n_a}} = \overbrace{\underbrace{[\mathbf{c}]}_{n}\underbrace{[\mathbf{0}]}_{n}\underbrace{[\mathbf{0}]}_{n_b}\underbrace{[\mathbf{0}]}_{n_a}}^{c2^{n+n_a+n_b}} + \overbrace{\underbrace{[\blacksquare\blacksquare]}_{n_b}\underbrace{[\blacksquare\blacksquare]}_{n_a}}^{c_f} \tag{2.8}$$

The black boxes represent generic non-zero bits. Here, we can see that the last n_a bits of the 2nd term on the rhs of Eq. (2.8) are required to be zero. Therefore, additional accuracy padding on a

gives us no more significant digits in our solution of b and the problem reduces to:



(2.9)

We are now ready to organize the main result of this section in the following theorem.

Theorem 1. *Given a scalar inversion problem with a mantissa of size n, the number of floating bits, n_b, necessary to invert the input is at most n.*

Proof. Our original problem has now been cast as one between integers $a \times \hat{b} = \hat{c}$, seen in Eq. 2.9. Assuming $a \neq 0$ (which, in our method, it is not by construction) by Euclidean division we are guaranteed that $c2^{n_b+n} = aq + r$ where $q, r \in \mathbb{Z}$ and $0 \leq r < |a|$.

$$\begin{aligned} a\hat{b} &= \hat{c} \\ a(b2^{n_b} + b_f) &= c2^{n_b+n} + c_f \\ a(\underbrace{b2^{n_b} + b_f}_{q}) \underbrace{-c_f}_{r} &= aq + r \end{aligned} \tag{2.10}$$

Since $r < |a|$, and a has n bit length, one would need *at most* $n_b = n$ bits to represent the floating bits, and our final output is provided by the first n bits of q. □

We conclude that to recover n bits of precision in the numerical inverse, one would need to extend the register of b (and for consistency, also c) by n bits. By doing so, the scalar inversion problem becomes a problem in exact bitwise arithmetic, and thus a circuit which can be exactly satisfied and solved by a SOLC.

Numerical Simulations- We constructed the solution of numerical inversion by extending the factorization solution found in [TDV17a]. The modified circuit contains the extended registers of satisfiability bits and freely floating nodes attached to voltage-controlled differential current

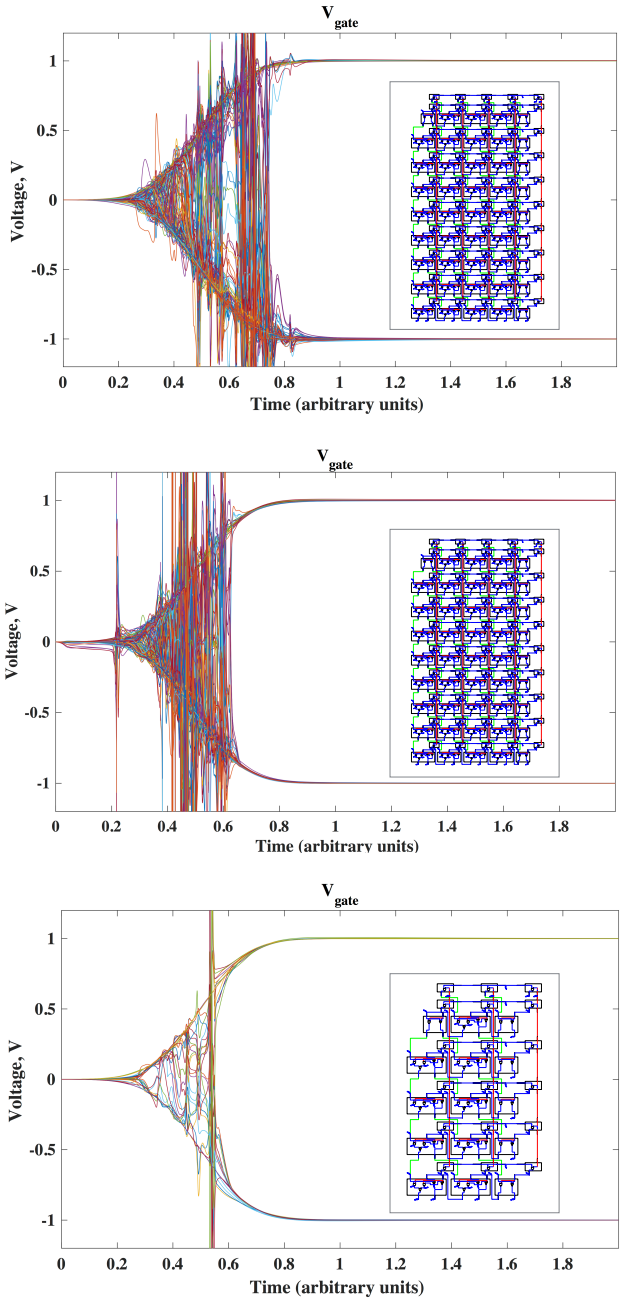


Figure 2.2: A plot of the voltage of all logic gate literals in several simulated systems with the corresponding circuit displayed as inset. The vertical axis represents voltages at the nodes with 1V representing the Boolean 1, and -1V the Boolean 0. The systems are found to be converged after some simulation time (arbitrary units). Plotted configurations are, from the top, inverting $a = 5$ in a 5-bit circuit, inverting $a = 3$ in a 5-bit circuit, and inverting $a = 3$ in a 3-bit circuit.

generators representing them. We performed simulations on up to 5-bit examples with different initial conditions using the Falcon© simulator[2].

A simplified circuit of a 2-bit inversion is shown in figure 2.1 with the internal logic gates inside the 2-bit and 3-bit adders shown for clarity. In figure 2.2 we plot the simulated voltages across several cases of scalar inversion. The topmost simulation is of a 5-bit circuit inverting $a = 10_{10} = 01010_2$ as a function of time, which converges to the logical 1 (voltage $V_{gate} = 1V$) and logical zero ($V_{gate} = -1V$) once the inverted solution is found. The solution is $b = 0.1_{10} = 0.00011001001100..._2$. While expressible in base 10 as a finite decimal, in binary the expression does not terminate. However, our circuit finds the correct 5 truncated digits of the exact solution $b \approx 0.00011_2$ in binary representation.

A few more examples are plotted for comparison. The simulation in the middle finds the inverse of $a = 3$ in a 5-bit circuit. Finally the bottom-most simulation finds the inverse of $a = 3$ in a 3 bit circuit.

Scaling- The multiplication operation on two n bit numbers involves n^2 AND gates and n additions. In our case, if we fix the number of bits of the input and increase the precision p of the inverse, the number of logic gates will scale as $O\left((n+p)^2\right)$. The resulting circuit scales quadratically in the number of input bits with fixed precision, and also quadratically in the precision, while fixing the number of bits of the input. Since the inversion is essentially an extended factorization, the equilibria will be reached exponentially fast and the total energy will scale polynomially in the amount of time to reach the equilibria as discussed in [TDV17a].

It is also worth noting that our inversion circuit performs the search of the equilibrium point collectively, by employing instantons [DVTO17]. This makes convergence times, if implemented in hardware, remain relatively independent from the system size since instantons are non-local objects that span the *entire* circuit irrespective of its size [DVTO17]. A demonstration of this

[2]Falcon© simulator is an in-house circuit simulator developed by F. L. Traversa optimized to simulate self-organizing circuits. Some examples of other simulations can be found, e.g., in [TDV17a, DVTO17, TPDV13, TB13, TB12]. In all cases, the resulting n bits of the inverse always matched the corresponding n bits of the exact answer.

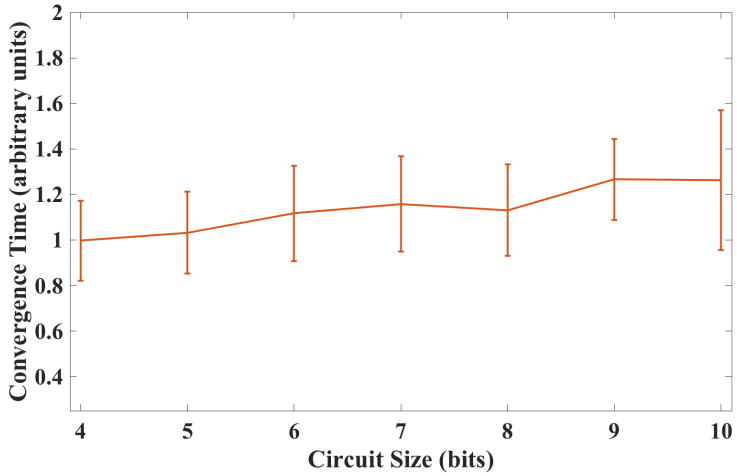


Figure 2.3: The average SOLC machine convergence times (proportional to the number of steps required to find the solution) across 10 simulations as a function of the size of the problem in bits. This plot shows the same scalar $(a = 2)$ being inverted with a variable circuit size. Since each simulation starts with random initial conditions, error bars around the mean are shown to give a sense of the resulting dispersion.

point is shown in Fig. 2.3, where we plot the machine time (namely the number of steps required to find the solution) versus the number of bits for a given inversion problem. We say the problem has converged when the maximum distance between all the voltages at the logic gates and true logical voltages (-1 and 1) is less than $\varepsilon = 0.01$. More precisely, the simulation has reached the convergence time, t_c, as soon as the following inequality is satisfied:

$$C(t_c) = \max_i \left\{ \min_{v_{\text{logical}} \in \{1,-1\}} |v_i(t_c) - v_{\text{logical}}| \right\} \leq \varepsilon \tag{2.11}$$

Here $v_i(t)$ represents the voltage at the i-th logic gate terminal in the circuit as a function of time. Taken first is the minimum over the logical voltages 1 and -1 which represent the Boolean 1 and 0 respectively. For clarity, we plot the function $C(t)$ in Fig. 2.4 together with the corresponding simulation. These numerical results give further evidence for the efficiency and scalability of the approach to numerical inversion we have developed here.

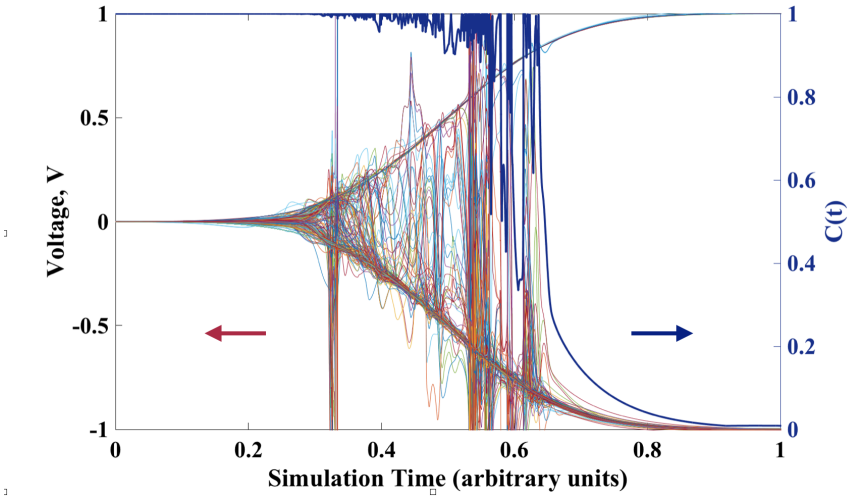


Figure 2.4: The function $C(t)$ [see Eq. (2.11)] and the related SOLC simulation for the case of 4-bit inversion of $a = 2$.

2.4 Extension to Matrix Inversion and Linear Systems

Once we have discussed explicitly the case of scalar inversion, it is now a simple (although cumbersome) exercise to extend it to the matrix inversion case. However, the explicit procedure for a general matrix inversion would require too many details to fit in this paper and we will report it in a subsequent publication. We then just provide the explicit procedure given a 2×2 non-singular matrix A.

Consider then the following matrix equation $AX = I$, where the matrix A is given with n bit binary entries, X is the solution we seek, and I is the identity matrix,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

This is equivalent to solving the following two linear systems of the form $Ax = b$, one

system for each column of the resulting inverse matrix.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The systems above are independent and can be solved in parallel. This gives us the following coupled equations that we must solve simultaneously with a SOLC:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2\,. \end{aligned} \tag{2.12}$$

There are 6 total arithmetical operations to be performed: 4 products and 2 sums. We approach the general inversion problem by constructing the relevant bit-wise logic circuit modules that will perform the fixed precision multiplication and addition. We perform all of the products in the manner we discussed above. The results of these four products are then summed (with signs) and set equal to b_1 and b_2 respectively.

The signed binary addition can be performed using the 2's complement method [Par09]. First, an XOR is applied to every non-sign bit of the products ($a_{11}x_1$, $a_{12}x_2$, etc.) with the sign bit (sign($a_{11}x_1$) ..). The sign bit is then added to the result of the XOR. This makes it such that if the product was negative, it takes the 2's complement (which is flipping every non-sign bit and adding 1 to the result) or if the product is positive, then the result is not modified. These 2's complement additions are applied to the outputs of all four products which occur in our 2×2 system. After which the resulting two additions are set equal to the 2's complements of b_1 and b_2.

This completes the SOLC for the linear system. The full inverse is found by applying this circuit to all columns of the given matrix.

2.5 Conclusion

We have demonstrated the power, and more importantly, the realizability, of self-organizing logic gates applied to scalar/matrix inversion and the solution of linear systems. The extensions and applications of this work are plentiful and far reaching. The method developed in the paper has direct applications and benefit to many fields (machine learning, statistics, signal processing, computer vision, engineering, optimization) that employ high-performance methods to solve linear systems. While the current work is immediately applicable to many problems, the concept can be extended to support IEEE floating point specification in the input and output for scientific computation.

The authors envision the effectiveness of these machines to be realized in specialized hardware that is designed to interface with current computers. These DMMs built in hardware will be capable of solving efficiently many numerical problems, scalar and matrix inversion being a small subset of potential applications which include matrix decompositions, eigenvalue problems, optimization problems, training models in machine learning, etc.

In analogy with how specialized GPUs interface with the standard CPUs of present computers to solve specific parallel problems much more efficiently than a CPU, a dedicated DMM can be constructed to work in tandem with current computers where the CPU/GPU would outsource computationally intensive problems which the DMM can solve rapidly. We thus hope our work will motivate experimental studies in this direction.

Acknowledgment

One of us (H.M.) acknowledges support from a DoD SMART scholarship. F.L.T. and M.D. acknowledge partial support from the Center for Memory and Recording Research at UCSD and LoGate Computing, Inc.

Chapter 2, in full, is a reprint of the material as it appears in IEEE transactions on neural networks and learning systems 2017 Volume 29. H. Manukian, F. L. Traversa and M. Di Ventra, "Memcomputing Numerical Inversion With Self-Organizing Logic Gates," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 6, pp. 2645-2650, June 2018, doi: 10.1109/TNNLS.2017.2697386. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Accelerating Deep Learning with Memcomputing

The following chapter was published as, Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. ”Accelerating deep learning with memcomputing.” Neural Networks 110 (2019): 1-7.

Abstract

Restricted Boltzmann machines (RBMs) and their extensions, often called “deep-belief networks”, are powerful neural networks that have found applications in the fields of machine learning and artificial intelligence. The standard way to training these models resorts to an iterative unsupervised procedure based on Gibbs sampling, called “contrastive divergence”, and additional supervised tuning via back-propagation. However, this procedure has been shown not to follow any gradient and can lead to suboptimal solutions. In this paper, we show an efficient alternative to contrastive divergence by means of simulations of digital memcomputing machines (DMMs) that compute the gradient of the log-likelihood involved in unsupervised training. We test our approach on pattern recognition using a modified version of the MNIST data set of hand-written numbers. DMMs sample very effectively the vast phase space defined by the probability distribution of

RBMs over the test sample inputs, and provide a very good approximation close to the optimum. This efficient search significantly reduces the number of generative pretraining iterations necessary to achieve a given level of accuracy in the MNIST data set, as well as a total performance gain over the traditional approaches. In fact, the acceleration of the pretraining achieved by *simulating* DMMs is comparable to, in number of iterations, the recently reported *hardware* application of the quantum annealing method on the same network and data set. Notably, however, DMMs perform far better than the reported quantum annealing results in terms of *quality* of the training. Finally, we also compare our method to recent advances in supervised training, like batch-normalization and rectifiers, that seem to reduce the advantage of pretraining. We find that the memcomputing method still maintains a quality advantage ($> 1\%$ in accuracy, corresponding to a 20% reduction in error rate) over these approaches, despite the network pretrained with memcomputing defines a more non-convex landscape using sigmoidal activation functions without batch-normalization. Our approach is agnostic about the connectivity of the network. Therefore, it can be extended to train full Boltzmann machines, and even deep networks at once.

3.1 Introduction

The progress in machine learning and big data driven by successes in deep learning is difficult to overstate. Deep learning models (a subset of which are called "deep-belief networks") are artificial neural networks with a certain amount of layers, n, with $n > 2$ [LBH15]. They have proven themselves to be very useful in a variety of applications, from computer vision [KSH12] and speech recognition [HDY$^+$12] to super-human performance in complex games[MKS$^+$15], to name just a few. While some of these models have existed for some time [Smo86], the dramatic increases in computational power combined with advances in effective training methods have pushed forward these fields considerably [Ben09].

Successful training of deep-belief models relies heavily on some variant of an iterative

gradient-descent procedure, called back-propagation, through the layers of the network [RHW86]. Since this optimization method uses only gradient information, and the error landscapes of deep networks are highly non-convex [CHM$^+$15], one would at best hope to find an appropriate local minimum.

However, there is evidence that in these high-dimensional non-convex settings, the issue is not getting stuck in some local minima but rather at saddle points, where the gradient also vanishes [DPG$^+$14], hence making the gradient-descent procedure of limited use. A takeaway from this is that a "good" initialization procedure for assigning the weights of the network, known as *pretraining*, can then be highly advantageous.

One such deep-learning framework that can utilize this pretraining procedure is the Restricted Boltzmann Machine (RBM) [Smo86], and its extension, the Deep Belief Network (DBN) [HOT06]. These machines are a class of neural network models capable of unsupervised learning of a parametrized probability distribution over inputs. They can also be easily extended to the supervised learning case by training an output layer using back-propagation or other standard methods [LBH15].

Training RBMs usually distinguishes between an unsupervised pretraining, whose purpose is to initialize a good set of weights, and the supervised procedure. The current most effective technique for pretraining RBMs utilizes an iterative sampling technique called *contrastive divergence* (CD) [Hin06]. Computing the exact gradient of the log-likelihood is exponentially hard in the size of the RBM, and so CD approximates it with a computationally friendly sampling procedure. While this procedure has brought RBMs most of their success, CD suffers from the slow mixing of Gibbs sampling, and is known not to follow the gradient of any function [ST10].

Partly due to these shortcomings of pretraining with CD, much research has gone into making the back-propagation procedure more robust and less sensitive to the initialization of weights and biases in the network. This includes research into different non-linear activation functions (e.g., "rectifiers") [GBB11] to combat the vanishing gradient problem and normalization

techniques (such as "batch-normalization") [IS15] that make back-propagation in deep networks more stable and less dependent on initial conditions. In sum, these techniques make training deep networks an easier (e.g., more convex) optimization problem for a gradient-based approach like back-propagation. This, in turn, relegates the standard CD pretraining procedure's usefulness to cases where the training set is sparse [LBH15], which is becoming an increasingly rare occurrence.

In parallel with this research into back-propagation, sizable effort has been expended toward improving the power of the pretraining procedure, including extensions of CD [TH09, RMDB18], CD done on memristive hardware [SRPJ15], and more recently, approaches based on quantum annealing that try to recover the exact gradient [BWP$^+$17] involved in pretraining. Some of these methods are classical algorithms simulating quantum sampling [WKS14], and still others attempt to use a *hardware* quantum device in contact with an environment to take independent samples from its Boltzmann distribution for a more accurate gradient computation. For instance, in a recent work, the state of the RBM has been mapped onto a commercial quantum annealing processor (a D-Wave machine), the latter used as a sampler of the model distribution [AH15]. The results reported on a reduced version of the well-known MNIST data set look promising as compared to CD [AH15]. However, these approaches require expensive hardware, and cannot be scaled to larger problems as of yet.

In the present paper, inspired by the theoretical underpinnings [TD15, TDV17a] and recent empirical demonstrations [TCSD18, MTD17] of the advantages of a new computing paradigm –*memcomputing* [DVP13a]– on a variety of combinatorial/optimization problems, we seek to test its power toward the computationally demanding problems in deep learning.

Memcomputing [DVP13a] is a novel computing paradigm that solves complex computational problems using processing embedded in memory. It has been formalized by two of us (FLT and MD) by introducing the concept of universal memcomputing machines[TD15]. In short, to perform a computation, the task at hand is mapped to a continuous dynamical system that

employs highly-correlated states [DVTO17] (in both space and time) of the machine to navigate the phase space efficiently and find the solution of a given problem as mapped into the equilibrium states of the dynamical system.

In this paper, we employ a subset of these machines called *digital memcomputing machines* (DMMs) and, more specifically, their self-organizing circuit realizations [TDV17a, DVT18]. The distinctive feature of DMMs is their ability to read and write the initial and final states of the machine *digitally*, namely requiring only *finite* precision. This feature makes them easily *scalable* as our modern computers.

From a practical point of view DMMs can be built with standard circuit elements with and without memory [TDV17a]. These elements, however, are *non-quantum*. Therefore, the ordinary differential equations of the corresponding circuits can be efficiently simulated on our present computers. Here, we will indeed employ only *simulations* of DMMs on a single Xeon processor to train RBMs. These simulations show already substantial advantages with respect to CD and even quantum annealing, despite the latter is executed on hardware. Of course, the hardware implementation of DMMs applied to these problems would offer even more advantages since the simulation times will be replaced by the actual physical time of the circuits to reach equilibrium. This would then offer a realistic path to real-time pretraining of deep-belief networks.

In order to compare directly with quantum annealing results recently reported[AH15], we demonstrate the advantage of our memcomputing approach by first training on a reduced MNIST data set as that used in Ref [AH15]. We show that our method requires far less pretraining iterations to achieve the same accuracy as CD, as well as an overall accuracy gain over both CD and quantum annealing. We also train the RBMs on the reduced MNIST data set without mini-batching, where the quantum annealing results are not available. Also in this case, we find both a substantial reduction in pretraining iterations needed as well as a higher level of accuracy of the memcomputing approach over the traditional CD.

Our approach then seems to offer many of the advantages of quantum approaches. How-

ever, since it is based on a completely classical system, it can be efficiently deployed in software (as we demonstrate in this paper) as well as easily implemented in hardware, and can be scaled to full-size problems.

Finally, we investigate the role of recent advances in supervised training by comparing accuracy obtained using only back-propagation with batch-normalization and rectifiers starting from a random initial condition versus the back-propagation procedure initiated from a network pretrained with memcomputing, *with* sigmoidal activations and *without* batch-normalization. Even without these advantages, namely operating on a more non-convex landscape, we find the network pretrained with memcomputing maintains an accuracy gain over state-of-the-art back-propagation by more than 1% and a 20% reduction in error rate. This gives further evidence to the fact that memcomputing pretraining navigates to an advantageous initial point in the non-convex loss surface of the deep network.

3.2 RBMs and Contrastive Divergence

An RBM consists of m visible units, $v_j, j = 1 \dots m$, each fully connected to a layer of n hidden units, $h_i, i = 1 \dots n$, both usually taken to be binary variables. In the restricted model, no intra-layer connections are allowed, see Fig. 3.1.

The connectivity structure of the RBM implies that, given the hidden variables, each input node is conditionally independent of all the others:

$$p(v_i, v_j|\mathbf{h}) = p(v_i|\mathbf{h})p(v_j|\mathbf{h}). \tag{3.1}$$

The joint probability is given by the Gibbs distribution,

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}, \tag{3.2}$$

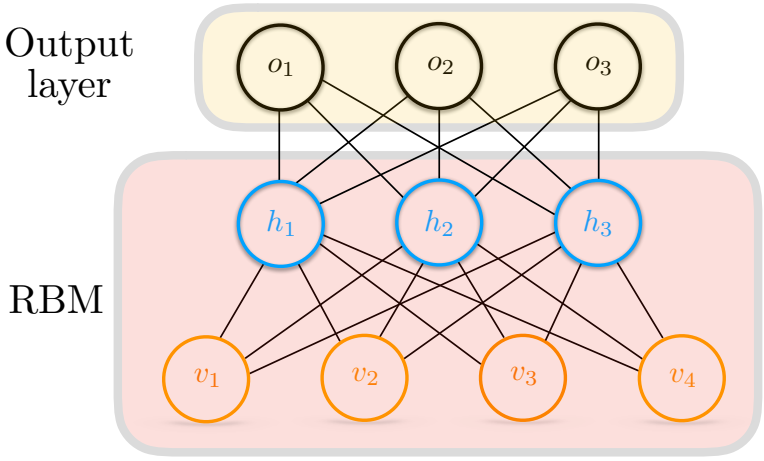


Figure 3.1: A sketch of an RBM with four visible nodes, three hidden nodes, and an output layer with three nodes. The value of each stochastic binary node is represented by $v_i, h_i \in \{0,1\}$, which are sampled from the probabilities in Eqs. (3.7), (3.8). The connections between the layers represent the weights, $w_{ij} \in \mathbb{R}$ (biases not shown). Note the lack of connections between nodes in the same layer, which distinguishes the RBM from a Boltzmann machine. The RBM weights are trained separately from the output layer with generative pretraining, then tuned together via back-propogation (just as in a feed-forward neural network).

with an energy function

$$E(\mathbf{v},\mathbf{h}) = -\sum_i \sum_j w_{ij} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i, \tag{3.3}$$

where w_{ij} is the weight between the i-th hidden neuron and the j-th visible neuron, and b_j, c_i are real numbers indicating the "biases" of the neurons. The value, Z, is a normalization constant, and is known in statistical mechanics as the partition function. Training an RBM then amounts to finding a set of weights and biases that maximizes the likelihood (or equivalently minimizes the energy) of the observed data.

A common approach to training RBMs for a supervised task is to first perform generative unsupervised learning (pretraining) to initialize the weights and biases, then run back-propagation

over input-label pairs to fine tune the parameters of the network. The pretraining is framed as a gradient ascent over the log-likelihood of the observed data, which gives a particularly tidy form for the weight updates from the n-th to the $(n+1)$-th iteration:

$$\Delta w_{ij}^{n+1} = \alpha \Delta w_{ij}^{n} + \varepsilon[\langle v_i h_j \rangle_{\text{DATA}} - \langle v_i h_j \rangle_{\text{MODEL}}], \tag{3.4}$$

where α is called the "momentum" and ε is the "learning rate". A similar update procedure is applied to the biases:

$$\Delta b_{i}^{n+1} = \alpha \Delta b_{i}^{n} + \varepsilon[\langle v_i \rangle_{\text{DATA}} - \langle v_i \rangle_{\text{MODEL}}], \tag{3.5}$$

$$\Delta c_{j}^{n+1} = \alpha \Delta c_{j}^{n} + \varepsilon[\langle h_j \rangle_{\text{DATA}} - \langle h_j \rangle_{\text{MODEL}}]. \tag{3.6}$$

This form of the weight updates is referred to as "stochastic gradient optimization with momentum". The first expectation value on the rhs of Eqs. (3.4), (3.5), and (3.6) is taken with respect to the conditional probability distribution with the data fixed at the visible layer. This is relatively easy to compute. Evaluation of the second expectation on the rhs of Eqs. (3.4), (3.5), and (3.6) is exponentially hard in the size of the network, since obtaining independent samples from a high-dimensional model distribution easily becomes prohibitive with increasing size [Hin06]. This is the term that CD attempts to approximate.

The CD approach attempts to reconstruct the difficult expectation term with iterative Gibbs sampling. This works by sequentially sampling each layer given the sigmoidal conditional probabilities, namely

$$p(h_i = 1|\mathbf{v}) = \sigma\left(\sum_j w_{ij} v_j + c_i\right), \tag{3.7}$$

for the visible layer, and similarly for the hidden layer

$$p(v_j = 1|\mathbf{h}) = \sigma\left(\sum_i w_{ij} h_i + b_j\right), \tag{3.8}$$

with $\sigma(x) = (1 + e^{-x})^{-1}$. The required expectation values are calculated with the resulting samples. In the limit of infinite sampling iterations, the expectation value is recovered. However, this convergence is slow and in practice usually only one iteration, referred to CD-1, is used [Hin10].

3.3 Efficient Sampling with Memcomputing

3.3.1 The memcomputing approach to optimization

In this work, we propose the application of DMMs to the accurate training of restricted Boltzmann machines. Formally, a DMM can be specified by the following tuple, [TDV17a]

$$\text{DMM} = (\mathbb{Z}_2, \Delta, \mathcal{P}, S, \Sigma, p_0, s_0, F) \tag{3.9}$$

Where Δ is the set of transition functions between states of the machine,

$$\delta_\alpha : \mathbb{Z}_2^{m_\alpha} \setminus F \times \mathcal{P} \to \mathbb{Z}_2^{m'_\alpha} \times \mathcal{P}^2 \times S \tag{3.10}$$

Here m_α represents the number of memprocessors read in by δ_α and m'_α the number of memprocessors written by the function. $\mathcal{P}$ is the set of arrays of pointers p_α that identify memprocessors called by δ_α, S is the collection of indices α, Σ is a set of initial states, p_0 an initial array of pointers, s_0 an initial index and F a set of final or accepting states.

A practical realization of DMMs can be accomplished using dynamical systems [TDV17a, DVT18]. In this paper, we employ the representation that uses self-organizing logic circuits (SOLCs), namely circuits that self-organize to satisfy the appropriate logical propositions defined by the problem at hand. These circuits are fully specified by a set of coupled ordinary differential equations representing the physical system of electric components comprising the circuit,

$$\dot{y} = G(y(t)), \tag{3.11}$$

where y describes the vector of all voltages, currents and internal state variables (providing memory) in the circuit, and G is the flow vector field that defines the laws of temporal evolution of the circuit.

To solve a given computational problem within this paradigm, first a Boolean circuit is constructed that represents the given problem. The constituent classical logic gates are then replaced with ones that self-organize [TDV17a]. The inputs to the problem can be specified via voltages at terminals of the SOLC, whose equations of motion are then integrated forward in time from a random initial condition, according to the vector flow field F, to an equilibrium (fixed) point of the system. Voltage terminals corresponding to the solution of the problem can then be read out.

Prima facie, the mapping of a discrete Boolean problem into a continuous system of non-linear differential equations may not seem computationally advantageous. In this case, however, the mapping to a SOLC results in a vector flow field which has been shown to possess certain functional and topological properties that give rise to dramatic computational advantages [TDV17a, DVT18]. Namely, the resulting dynamical systems do not exhibit chaos or periodic orbits in the presence of solutions [DVT17, DT17], and employ highly non-local (in both space and time) correlated states to efficiently explore the vast phase space of the problem [DVTO17, BMTD18]. It is the latter property that truly distinguishes this approach from other, local, approaches to optimization.

3.3.2 From RBMs to a QUBO problem to a MAX-SAT

To employ these advantages within the training of RBMs, we construct first a reinterpretation of the RBM pretraining that explicitly shows how it corresponds to an NP-hard optimization

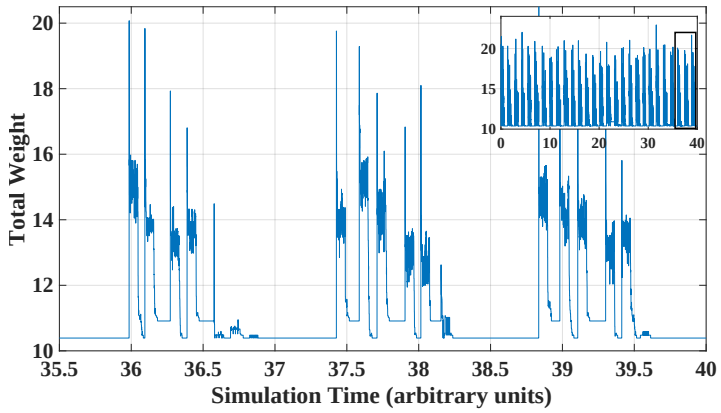


Figure 3.2: Plot of the total weight of a MAX-SAT clause as a function of internal simulation time (not physical seconds) of a DMM. A lower weight variable assignment corresponds directly to a higher probability assignment of the nodes of an RBM. If the simulation has not changed assignments in some time, we restart with another random (independent) initial condition. The inset shows the full simulation, with all restarts. The main figure focuses on the last three restarts, signified by the black box in the inset.

problem, which we then tackle using DMMs in their SOLC representation. We first observe that to obtain a sample near most of the probability mass of the joint distribution, $p(v,h) \propto e^{-E(v,h)}$, one must find the minimum of the energy of the form Eq. (3.3), which constitutes a *quadratic unconstrained binary optimization* (QUBO) problem [AB09].

We can see this directly by considering the visible and hidden nodes as one vector $\mathbf{x} = (\mathbf{v}, \mathbf{h})$ and re-writing the energy of an RBM configuration as

$$E = -x^T Q x, \tag{3.12}$$

where Q is the matrix

$$Q = \begin{bmatrix} B & W \\ 0 & C \end{bmatrix}, \tag{3.13}$$

with B and C being the diagonal matrices representing the biases b_j and c_i, respectively, while the matrix W contains the weights.

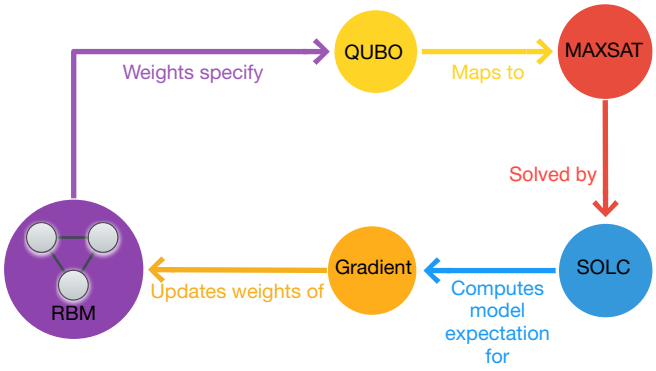


Figure 3.3: A diagrammatic overview of the memcomputing-assisted RBM training procedure described in this paper. For each iteration of pretraining, the set of weights of an RBM specifies a QUBO problem, which is converted into an equivalent weighted MAX-SAT problem to be solved by a DMM in its SOLC representation. This solution computes the model expectation term in the gradient (3.4), which updates the weights of the model and the whole process begins again, until convergence.

We then employ a mapping from a general QUBO problem to a *weighted maximum satisfiability* (weighted MAX-SAT) problem, similar to [BCMR10], which is directly solved by the DMM. The weighted MAX-SAT problem is to find an assignment of boolean variables that minimizes the total weight of a given boolean expression written in conjunctive normal form [AB09].

This problem is a well-known problem in the NP-hard complexity class [AB09]. However, it was recently shown in Ref. [TCSD18], that simulations of DMMs show dramatic (exponential) speed-up over the state-of-the-art solvers, when attempting to find better approximations to hard MAX-SAT instances beyond the inapproximability gap [Hås01]. We then use SOLCs appropriately designed to tackle the MAX-SAT that originates from the RBM QUBO problem. The ordinary differential equations we solve can be found in Ref. [TDV17a] appropriately adapted to deal with the particular problem discussed in this paper.

The approximation to the global optimum of the weighted MAX-SAT problem given

by memcomputing is then mapped back to the original variables that represent the states of the RBM nodes. Finally, we obtain an approximation to the "ground state" (lowest energy state) of the RBM as a variable assignment, $\mathbf{x}^*$, close to the peak of the probability distribution, where $\nabla P(\mathbf{x}^*) = 0$. This assignment is obtained by integration of the ordinary differential equations that define the SOLC's dynamics. In doing so we collect an entire trajectory, $\mathbf{x}(t)$, that begins at a random initial condition in the phase space of the problem, and ends at the lowest energy configuration of the variables (see Fig. 3.2).

Since the problem we are tackling here is an optimization one, we do not have any guarantee of finding the *global* optimum. (This is in contract to a SAT problem where we can guarantee DMMs *do* find the solutions of the problem corresponding to equilibrium points, if these exist [TDV17a, DT17, DVT17].) Therefore, there is an ambiguity about what exactly constitutes the stopping time of the simulation, since *a priori*, one cannot know that the simulation has reached the *global* minimum.

We then perform a few "restarts" of the simulation (that effectively correspond to a change of the initial conditions) and stop the simulation when the machine has not found any better configuration within that number of restarts. The restarts are clearly seen in Fig. 3.2 as spikes in the total weight of the boolean expression. In this work we have employed 28 restarts, which is an over-kill since a much smaller number would have given similar results.

The full trajectory, $\mathbf{x}(t)$, together with the above "restarts" is plotted in Fig. 3.2. It is seen that this trajectory, in between restarts, spends most of its time in "low-energy regions," or equivalently areas of high probability. A time average, $\langle \mathbf{x}(t) \rangle$, gives a good approximation to the required expectations in the gradient calculation in Eqs. (3.4), (3.5), and (3.6). In practice, even using the best assignment found, $\mathbf{x}^*$, shows a great improvement over CD in our experience. This is what we report in this paper. Note also that a full trajectory, as the one shown in Fig. 3.2, takes about 0.5 seconds on a single Xeon processor. A schematic of the full pre-training iteration procedure used in this paper can be seen in the diagram in Fig. 3.3.

3.4 Results

As a testbed for the memcomputing advantage in deep learning, and as a direct comparison to the quantum annealing hardware approaches, we first looked to the reduced MNIST data set as reported in [AH15] for quantum annealing using a D-wave machine. Therefore, we have first applied the same reduction to the full MNIST problem as given in that work, which consists of removing two pixels around all 28×28 grayscale values in both the test and training sets. Then each 4×4 block of pixels is replaced by their average values to give a 6×6 reduced image. Finally, the four corner pixels are discarded resulting in a total of 32 pixels representing each image.

We also trained the same-size DBN consisting of two stacked RBMs each with 32 visible and hidden nodes, training each RBM one at a time. We put both the CD-1 and our Memcomputing-QUBO (Mem-Qubo) approach through $N = 1, \cdots, 50$ generative pretraining iterations using no mini-batching.

For the memcomputing approach, we solve one QUBO problem per pretraining iteration to compute the model expectation value in Eqs. (3.4), (3.5), and (3.6). We pick out the best variable assignment, $\mathbf{x}^*$, which gives the ground state of Eq. (3.3) as an effective approximation of the required expectation. After generative training, an output classification layer with 10 nodes was added to the network (see Fig. 3.1) and 1000 back-propagation iterations were applied in both approaches using mini-batches of 100 samples to generate Fig. 3.4. For both pretraining and back-propagation, our learning rate was set to $\varepsilon = 0.1$ and momentum parameters were $\alpha = 0.1$ for the first 5 iterations, and $\alpha = 0.5$ for the rest, same as in [AH15].

Accuracy on the test set versus CD-1 as a function of the number of pretraining iterations is seen in Fig. 3.4. The memcomputing method reaches a far better solution faster, and maintains an advantage over CD even after hundreds of back-propagation iterations. Interestingly, our *software* approach is even competitive with the quantum annealing method done in *hardware*

[AH15] (cf. Fig. 3.4 with Figs. 7, 8, and 9 in Ref. [AH15]). This is quite a remarkable result, since we integrate a set of differential equations of a *classical* system, in a scalable way, with comparable sampling power to a physically-realized system that takes advantage of *quantum* effects to improve on CD.

Finally, we also trained the RBM on the reduced MNIST data set without mini-batches. We are not aware of quantum-annealing results for the full data set, but we can still compare with the CD approach. We follow a similar procedure as discussed above. In this case, however, no mini-batching was used for a more direct comparison between the Gibbs sampling of CD and our memcomputing approach. The results are shown in Fig. 3.5 for different numbers of back-propagation iterations. Even on the full modified MNIST set, our memcomputing approach requires a substantially lower number of pretraining iterations to achieve a high accuracy and, additionally, shows a higher level of accuracy over the traditional CD, even after 800 back-propagation iterations.

3.5 The Role of Supervised Training

The computational difficulty of computing the exact gradient update in pretraining, combined with the inaccuracies of CD, has inspired research into methods which reduce (or outright eliminate) the role of pretraining deep models. These techniques include changes to the numerical gradient procedure itself, like adaptive gradient estimation [KB14], changes to the activation functions (e.g., the introduction of rectifiers) to reduce gradient decay and enforce sparsity [GBB11], and techniques like batch normalization to make back-propagation less sensitive to initial conditions [IS15]. With these new updates, in many contexts, deep networks initialized from a random initial condition are found to compete with networks pretrained with CD [GBB11].

To complete our analysis we have then compared a network pretrained with our memcomputing approach to these back-propagation methods with no pretraining. Both networks

were trained with stochastic gradient descent with momentum and the same learning rates and momentum parameter we used in Section 3.3.

In Fig. 3.6, we see how these techniques fair against a network pretrained with the memcomputing approach on the reduced MNIST set. In the randomly initialized network, we employ the batch-normalization procedure [IS15] coupled with rectified linear units (ReLUs) [GBB11]. As anticipated, batch-normalization smooths out the role of initial conditions, while rectifiers should render the energy landscape defined by Eq. (3.3) more convex. Therefore, combined they indeed seem to provide an advantage compared to the network trained with CD using sigmoidal functions [GBB11]. In fact, in experiments with batch normalization and ReLUs, we did not find any accuracy difference between a network pre-trained with CD and a randomly initialized network, and thus we omit plotting CD results in Fig. 3.6.

However, they are not enough to overcome the advantages of our memcomputing approach. In fact, it is obvious from Fig. 3.6 that the network pretrained with memcomputing maintains an accuracy advantage (of more than 1% and a 20% reduction in error rate) on the test set out to more than a thousand back-propagation iterations. It is key to note that the network pretrained with memcomputing contains sigmoidal activations compared to the rectifiers in the network with no pretraining. Also, the pretrained network was trained without any batch normalization procedure.

Therefore, considering all this, the pretrained network should pose a more difficult optimization problem for stochastic gradient descent. Instead, we found an accuracy advantage of memcomputing throughout the course of training. This points to the fact that with memcomputing, the pretraining procedure is able to operate close to the "true gradient" (Eqs. (3.4), (3.5), and (3.6)) during training, and in doing so, initializes the weights and biases of the network in a advantageous way.

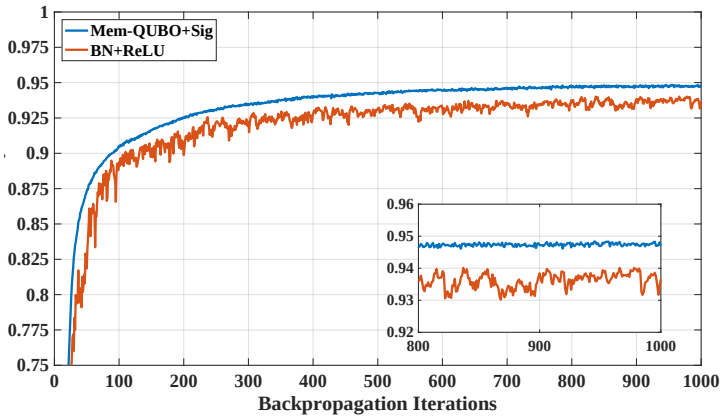


Figure 3.6: Accuracy on the reduced MNIST test set obtained on a network pretrained with (blue curve) our approach Mem-QUBO and sigmoidal activation functions (Sig) versus the same size network with (red curve) no pretraining but with batch normalization (BN) and rectified linear units (ReLU). Both networks were trained with stochastic gradient descent with momentum and mini-batches of 100. The inset clearly shows an accuracy advantage of Mem-QUBO greater than 1% and an error rate reduction of 20% throughout the training.

3.6 Conclusions

In this paper we have demonstrated how the memcomputing paradigm (and, in particular, its digital realization [TDV17a]) can be applied toward the chief bottlenecks in deep learning today. In this paper, we directly assisted a popular algorithm to pretrain RBMs and DBNs, which consists of gradient ascent on the log-likelihood. We have shown that memcomputing can accelerate considerably the pretraining of these networks toward better quality solutions far better than what is currently done.

In fact, *simulations* of digital memcomputing machines achieve accelerations of pretraining comparable to, in number of iterations, the *hardware* application of the quantum annealing method, but with better quality. In addition, unlike quantum computers, our approach can be easily scaled on classical hardware to full size problems. In fact, the method we employ to solve the MAX-SAT has been shown to scale to tens of millions of variables (as opposed to only hundreds to thousands encountered in this paper) [TCSD18]. We leave the study of other full size

problems for future work.

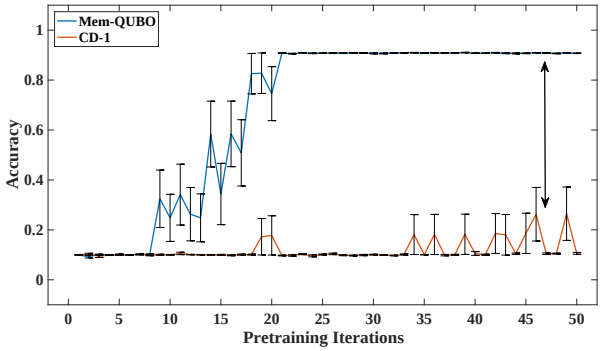
In addition, our memcomputing method retains an advantage also with respect to advances in supervised training, like batch-norming and rectifiers, that have been introduced to eliminate the need of pretraining. We find indeed, that despite our pretraining done with sigmoidal functions, hence on a more non-convex landscape than that provided by rectifiers, we maintain an accuracy advantage greater than 1% (and a 20% reduction in error rate) throughout the training.

Finally, the form of the energy in Eq. (3.3) is quite general and encompasses full DBNs. In this way, our method can also be applied to pretraining *entire* deep-learning models at once, potentially exploring parameter spaces that are inaccessible by any other classical or quantum methods. We leave this interesting line of research for future studies.

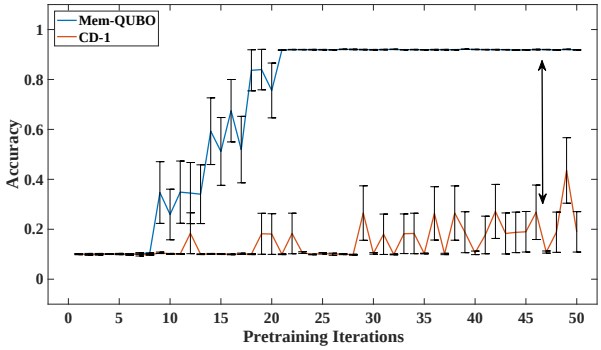
Acknowledgments – We thank Forrest Sheldon for useful discussions and Yoshua Bengio for pointing out the role of supervised training techniques. H.M. acknowledges support from a DoD-SMART fellowship. M.D. acknowledges partial support from the Center for Memory and Recording Research at UCSD. All memcomputing simulations reported in this paper have been done using the Falcon© simulator of MemComputing, Inc. (http://memcpu.com/).

Chapter 3, in full, is a reprint of the material as it appears in, Manukian, Haik, Fabio L. Traversa, and Massimiliano Di Ventra. "Accelerating deep learning with memcomputing." Neural Networks 110 (2019): 1-7. The dissertation author was the primary investigator and author of this paper.

(a) 100 back-propagation iterations



(b) 200 back-propagation iterations



(c) 400 back-propagation iterations

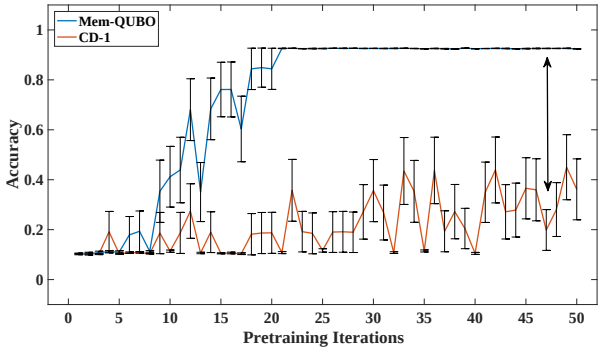
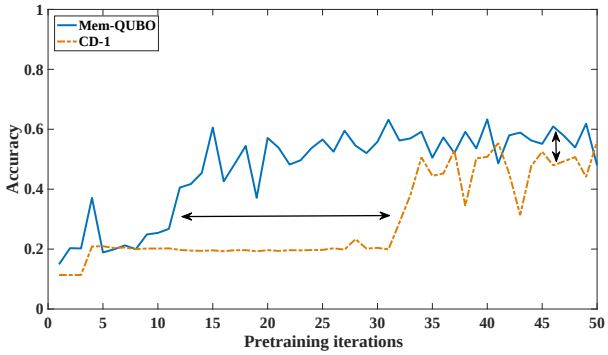
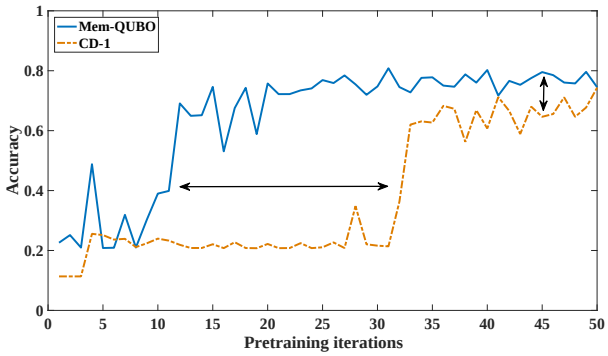


Figure 3.4: Memcomputing (Mem-QUBO) accuracy on the test set of the reduced MNIST problem versus contrastive divergence for $n = 100(a), 200(b), 400(c)$ iterations of back-propagation with mini-batches of 100. The plots show average accuracy with $\pm\sigma/\sqrt{N}$ error bars calculated across 10 DBNs trained on $N = 10$ different partitions of the training set. One can see a dramatic acceleration with the memcomputing approach needing far less iterations to achieve the same accuracy, as well as an overall performance gap (indicated by a black arrow) that back-propagation cannot seem to overcome. Note that some of the error bars for both Mem-QUBO and CD-1 are very small on the reported scale for a number of pretraining iterations larger than about 20.

(a) 100 back-propagation iterations



(b) 500 back-propagation iterations



(c) 800 back-propagation iterations

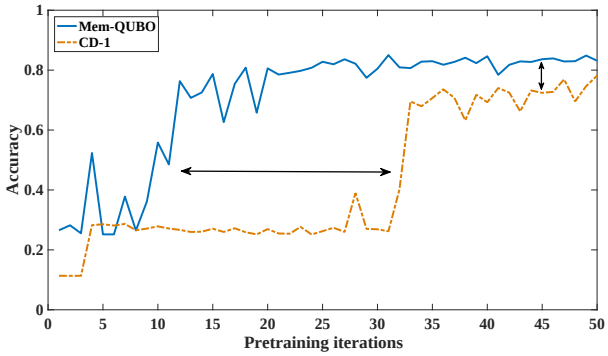


Figure 3.5: Mem-QUBO accuracy on the reduced MNIST test set vs. CD-1 after $n = 100(a), 500(b), 800(c)$ iterations of back-propagation with no mini-batching. The resulting pretraining acceleration shown by the memcomputing approach is denoted by the horizontal arrow. A performance gap also appears, emphasized by the vertical arrow, with Mem-QUBO obtaining a higher level of accuracy than CD-1, even for the highest number of back-propagation iterations. No error bars appear here since we have trained the full test set.

Chapter 4

Mode-Assisted Unsupervised Learning of Restricted Boltzmann Machines

The following chapter was published as Manukian, Haik, Yan Ru Pei, Sean RB Bearden, and Massimiliano Di Ventra. "Mode-assisted unsupervised learning of restricted Boltzmann machines." Communications Physics 3, no. 1 (2020): 1-8.

Abstract

Restricted Boltzmann machines (RBMs) are a powerful class of generative models, but their training requires computing a gradient that, unlike supervised backpropagation on typical loss functions, is notoriously difficult even to approximate. Here, we show that properly combining standard gradient updates with an off-gradient direction, constructed from samples of the RBM ground state (mode), improves training dramatically over traditional gradient methods. This approach, which we call 'mode-assisted training', promotes faster training and stability, in addition to lower converged relative entropy (KL divergence). We demonstrate its efficacy on synthetic datasets where we can compute KL divergences exactly, as well as on a larger machine learning standard (MNIST). The proposed mode-assisted training can be applied in conjunction with any given gradient method, and is easily extended to more general energy-based neural

network structures such as deep, convolutional and unrestricted Boltzmann machines.

4.1 Introduction

Boltzmann machines [AHS85] and their restricted version (RBMs), are unsupervised generative models applied to a variety of machine learning problems [GBCB16]. They enjoy a universal approximation theorem for discrete probability distributions [LRB08], are used as building blocks for deep-belief networks [Ben09] and, in no small feat, can even represent correlated states in quantum many-body systems [CT17, GD17]. Despite these advantages, the limitations of the most popular training algorithms for RBMs, combined with rapid advances in supervised learning techniques, have led to the sideline of their unsupervised learning, known also as "pretraining", in favor of supervised backpropagation from random initial conditions [Ben09].

However, supervised learning requires large datasets of labeled examples, and even then, state-of-the-art neural networks have been shown to be vulnerable to what are called adversarial examples [SZS$^+$13], or slight perturbations of the input that 'fool' the network. On the other hand, unsupervised pretraining is possible in the absence of labels, and is known to be a strong regularizer [EBC$^+$10], often resulting in better generalization for supervised models. Since adversarial vulnerability could be seen as a failure in generalization, an improvement in unsupervised training could lead to more robust performance in a downstream task. This motivates the search for better unsupervised training methods.

To set the stage for our training method, we provide a short introduction to RBMs and then describe standard training approaches and their limitations. RBMs are undirected weighted graphs with a bipartite structure that differentiates between n visible nodes, $\mathbf{v} \in \{0,1\}^n$ and m latent, or 'hidden', nodes, $\mathbf{h} \in \{0,1\}^m$, not directly constrained by the data [GBCB16]. (Note that an RBM does not allow connections within a layer.) These states are usually taken to be binary

but can be generalized. Each state of the machine corresponds to an energy of the form

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h}, \tag{4.1}$$

where the biases $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and weights $\mathbf{W} \in \mathbb{R}^{n\times m}$ are the learnable parameters. This induces a distribution over states given by a Boltzmann-Gibbs distribution,

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\mathcal{Z}}. \tag{4.2}$$

The normalizing factor, $\mathcal{Z} = \sum_{\{\mathbf{v}\}}\sum_{\{\mathbf{h}\}} e^{-E(\mathbf{v},\mathbf{h})}$, is the formidable partition function that involves the sum over an exponentially scaling number of states, making the exact computation of its value infeasible. Additionally, the bipartite structure of the RBM makes the hidden nodes conditionally independent given the visible nodes (and vice versa), with a closed form conditional distribution given by [Hin02] $p(h_j = 1 \,|\, \mathbf{v}) = \sigma(\sum_i w_{ij}v_i + b_j)$, where $\sigma(x) = (1+e^{-x})^{-1}$.

RBMs can learn to represent some unknown data distribution over the visible layer, $q(\mathbf{v})$, typically specified by a data set of samples. We indicate the unique elements of this data set as $\mathcal{D} = \{\mathbf{v}_1, \cdots, \mathbf{v}_{n_d}\} \subset \Omega$, where Ω is the space of all binary sequences of length n. Let us assume further that all data points have equal amplitude over the support, i.e., $p_i = 1/n_d$. Since most real world datasets consist of unique elements with no exact repeats, this class of distributions includes all relevant ones.

The RBM is tasked to match its marginal distribution over the visible layer, $p(\mathbf{v}) = \sum_{\{\mathbf{h}\}} p(\mathbf{v},\mathbf{h})$, to this unknown data distribution. The former can be written as,

$$p(\mathbf{v}) = \frac{1}{\mathcal{Z}}\prod_{i=1}^{n} e^{a_i v_i} \prod_{j=1}^{m}\left(1 + e^{b_j + \sum_{i'=1}^{n} w_{i'j}v_{i'}}\right). \tag{4.3}$$

Training an RBM then amounts to a search for the appropriate weights and biases that will minimize the quantity known as the Kullback-Leibler (KL) divergence between the two

distributions,

$$\mathrm{KL}(q||p) = \sum_{\{\mathbf{v}\}} q(\mathbf{v}) \log \frac{q(\mathbf{v})}{p(\mathbf{v})}. \tag{4.4}$$

The optimization of Eq. (5.3) is typically done via stochastic gradient descent with respect to the RBM parameters, which leads to weight updates of the form [FI12],

$$\Delta w_{ij} \propto \left[\langle v_i h_j \rangle_{q(\mathbf{v}) p(\mathbf{h}\,|\,\mathbf{v})} - \langle v_i h_j \rangle_{p(\mathbf{v},\mathbf{h})} \right]. \tag{4.5}$$

The first term on the rhs of Eq. (5.4) is an expectation with the hidden nodes driven by the data, and is referred to as the "data term". Since the conditional distributions across the hidden nodes are factorial and known in closed form, this inference problem is easy in the RBM case. The second term on the rhs of Eq. (5.4), instead, is an expectation over the model distribution with no nodes fixed, and called the "model term". The exact calculation of this term requires computing the partition function of the RBM, which is proved to be hard even to estimate [LS10].

This expectation value is popularly approximated by a Markov Chain Monte Carlo (MCMC) procedure dubbed 'contrastive divergence' (CD) [Hin02]. Since CD initializes Markov chains from elements of the dataset, difficulties arise when the model distribution represented by the RBM contains modes where the data distribution has negligible probability, sometimes referred to as 'spurious modes'. The prohibitively slow mixing due to random-walk exploration, and typical high dimensionality of the problem render it difficult for CD to find and correct the probability of these modes.

In a recent work, a memcomputing-assisted training scheme for RBMs [MTDV19] was proposed to address this training difficulty. Memcomputing [DVP13b] is a novel computing paradigm in which memory (time non-locality) assists in the solution of hard decision and optimization problems [TDV17b, DT18]. In that previous work [MTDV19], the difficult model expectation in Eq. (5.4) was replaced by a sample of the RBM mode obtained by a memcomputing solver, which led to better quality solutions versus CD in a downstream classification task.

However, despite demonstrating a significant reduction in the number of pretraining iterations necessary to achieve a given level of classification accuracy, as well as a total performance gain over CD, that algorithm [MTDV19] does not fully exploit samples of the mode. In particular, it does not give rise to training advantages over standard methods in the unsupervised setting.

In the present work, we overcome this limitation. We show that by appropriately combining the RBM's mode (ground state) samples and data initiated chains (as in CD) not only improves considerably the model quality over CD and other MCMC procedures, but also improves the stability of the pre-training routine. Specifically, we introduce *i*) a principled schedule for incorporating samples of the RBM ground state into pre-training, *ii*) an appropriate mode-driven learning rate, *iii*) show comparisons to other state-of-the-art unsupervised pre-training approaches without the need of supervised fine-tuning, and *iv*) provide proofs of advantageous properties of the method. To corroborate our method, we will show exact KL/log-likelihood achieved on small synthetic datasets and on the MNIST dataset. In all cases, we find that mode-assisted training is able to learn more accurate models than several other training methods such as CD, persistent contrastive divergence (PCD), parallel tempering (PT) used in tandem with enhanced gradient RBMs (E-RBMs), and centered RBMs (C-RBMs) [MFW16].

4.2 Results

After these preliminaries we can now describe our mode-assisted training method. It consists in replacing the average in the model term of Eq. (5.4) with the mode of $p(\mathbf{v})$ at appropriate steps of the training procedure. However, $p(\mathbf{v})$ is very cumbersome to compute (see Eq. (4.3)), thereby adding a considerable computational burden. Instead, we sample the *mode* of $p(\mathbf{v},\mathbf{h})$, the model distribution of the RBM.

The rationale for replacing the mode, $\mathbf{v}^+$, of $p(\mathbf{v})$ with the visible configuration of the mode, $\mathbf{v}^*$, of $p(\mathbf{v},\mathbf{h})$ is because the two modes are equivalent with high probability under scenarios

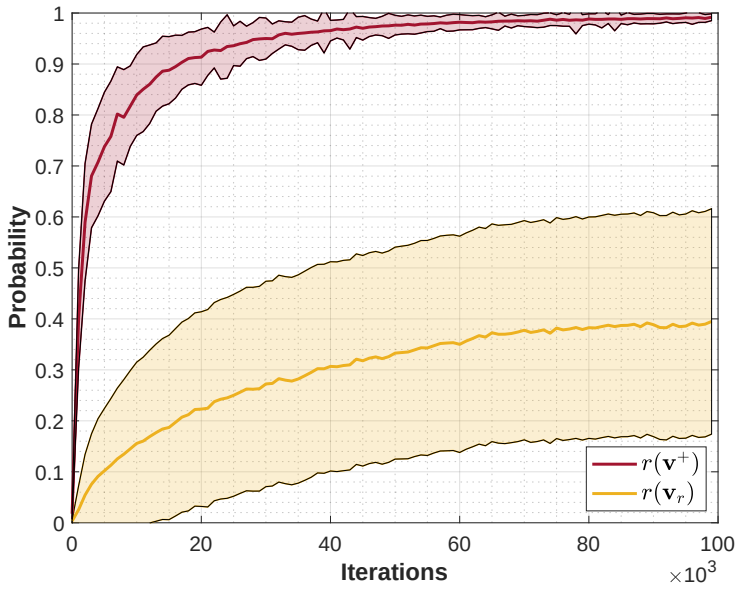


Figure 4.1: Maximal Conditional Probability During Training. We plot, in red, the maximal conditional probability of the hidden layer, $r(\mathbf{v}^+)$, where $\mathbf{v}^+$ is the mode of the marginal distribution, $p(\mathbf{v})$, as a function of contrastive divergence training iterations ($k = 1$). The results are averaged over an ensemble of 200 randomly initialized 15×10 Restricted Boltzmann Machines trained on a shifting bar dataset, with $\pm 1\sigma$ defining the shaded regions, where σ is the standard deviation. The same calculation conditioned on a random visible configuration, $\mathbf{v}_r$, is plot as a baseline for comparison in yellow.

typical for different stages of the RBM pre-training. We prove this in the Supplementary Note 1, while here we provide numerical evidence of this fact.

4.2.1 Mode Correspondence of Joint and Marginal Distributions

To illustrate the equivalence between the modes of $p(\mathbf{v})$ and $p(\mathbf{v},\mathbf{h})$, let us begin by expressing the joint probability mass function (PMF) in terms of the product of the marginal PMF over the visible layer and the conditional PMF over the hidden layer $p(\mathbf{v},\mathbf{h}) = p(\mathbf{v})p(\mathbf{h}\,|\,\mathbf{v})$. For any given visible configuration $\mathbf{v}$, we then have $\max_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) = p(\mathbf{v})\left[\max_{\mathbf{h}} p(\mathbf{h}\,|\,\mathbf{v})\right]$. We can then define the hidden "activation" of $\mathbf{v}$ to be

$$r(\mathbf{v}) = \max_{\mathbf{h}} p(\mathbf{h}\,|\,\mathbf{v}), \tag{4.6}$$

which allows us to write $\max_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) = p(\mathbf{v})r(\mathbf{v})$. Note that we can interpret $r(\mathbf{v})$ as a measure of the "certainty" that the hidden nodes acquire the value 0 or 1.

It is then clear that we can write the probability amplitude of the mode of the joint PMF as $\max_{\{\mathbf{v},\mathbf{h}\}} p(\mathbf{v},\mathbf{h}) = \max_{\mathbf{v}}(\max_{\mathbf{h}} p(\mathbf{v},\mathbf{h})) = \max_{\mathbf{v}}(p(\mathbf{v})r(\mathbf{v})) \leq \max_{\mathbf{v}}(p(\mathbf{v})) = p(\mathbf{v}^+)$, where we have used the fact that $r(\mathbf{v}) \leq 1$ and $\mathbf{v}^+$ is the mode of the marginal PMF, $p(\mathbf{v})$. If $r(\mathbf{v}^+) = 1$ then we have modal equivalence of the joint and marginal PMFs.

In Fig. 4.1, we plot the evolution of $r(\mathbf{v}^+)$ as a function of the number of CD-1 training iterations for a shifting bar synthetic dataset, which is small enough that we can compute the exact mode of $p(\mathbf{v})$ at any iteration. The figure indeed shows that $r(\mathbf{v}^+)$ approaches 1 rather quickly as pre-training proceeds. The activation of a random visible configuration is being used as comparison.

In the Supplementary Note 1 we also prove that the condition of $r(\mathbf{v}^+)$ being close to 1 is not necessary for establishing modal equivalence. In fact, we prove that it is still possible for the two modes to be equal even when the weights are small (thus a smaller $r(\mathbf{v}^+)$ value). Additionally, we show in the Supplementary Note 1 that mode-assisted training is more effective in exploring the PMF of the model distribution for RBM instances of greater frustration. The latter is a measure of the degeneracy of the low-energy states of an RBM, and thus the difficulty of finding the ground state configuration. Since it was shown that the frustration of the RBM increases as pre-training proceeds [PMDV19], in order to effectively utilize the power of mode-assisted training, the frequency of mode updates should be higher at the later stages of the training than the earlier stages.

4.2.2 Optimal Mode-Training Schedule

The results from the previous subsection then suggest a schedule for the mode-assisted training routine that performs mode updates more frequently the longer the pre-training routine has elapsed.

To realize this, we use a sigmoid, σ, to calculate the probability of replacing the data driven hidden CD term with a mode driven term at the iteration step n

$$P_{\text{mode}}(n) = P_{\max}\sigma(\alpha n + \beta). \tag{4.7}$$

Here, $0 < P_{\max} \leq 1$ is the maximum probability of employing a mode update, and α and β are parameters that control how the mode updates are introduced into the pre-training. They are chosen such that the frequency of mode updates becomes dominant only when both the conditions of large weights and frustration are met (see the "Methods" section for the value of these parameters). Initially, P_{mode} will be small, since the joint- and marginal-distribution modes are unequal, and gradually rises to its maximal value when the modes are of equal magnitude. Note that one may employ different functions to quantify the degree to which the joint- and marginal-distribution modes equalize during training. However, we have found that the sigmoid works well enough in practice.

4.2.3 Combining Markov Chains with Mode Updates

We are now ready to outline the full procedure of mode-assisted training, that combines a MCMC method with the mode updates following the schedule (4.7). Although one may choose any variation of the MCMC method to train RBMs, for definiteness of discussion, we consider here the standard training method, CD [Hin02]. In this case, weight updates follow the modified $\text{KL}(q||p)$ gradient. As discussed in the introduction, it evaluates to a difference of two expectations called the data term and model term which we can write as

$$\Delta w_{ij}^{\text{CD}} = \varepsilon^{\text{CD}} \left[\langle v_i h_j \rangle_{q(\mathbf{v})p(\mathbf{h}|\mathbf{v})} - \langle v_i h_j \rangle_{p^k(\mathbf{v},\mathbf{h})} \right], \tag{4.8}$$

where ε^{CD} is the CD update learning rate, and the expectation in the second term is taken over the reconstructed distribution over a Markov chain initialized from the dataset after k Gibbs samples

($k = 1$ in most cases). When driving the weights with samples of the RBM ground state with the schedule (4.7), we use instead the following update,

$$\Delta w_{ij}^{\text{mode}} = \varepsilon^{\text{mode}} \left[\langle v_i h_j \rangle_{q(\mathbf{v})p(\mathbf{h}\,|\,\mathbf{v})} - [v_i h_j]_{p(\mathbf{v},\mathbf{h})} \right], \tag{4.9}$$

where $[\,]_p$ is the mode of the joint RBM model distribution. Note that the mode update learning rate, $\varepsilon^{\text{mode}}$, may be different from the CD learning rate, ε^{CD}.

We also stress that the updates in Eq. (4.9) are in an off-gradient direction. As we show now, this is the reason for the increased stability of the training over MCMC approaches, and its convergence to arbitrarily small KL divergences.

4.2.4 Stability and Convergence

The data term, which is identical in both Eq. (4.8) and Eq. (4.9), tends to increase the weights associated with the visible node configurations in the dataset, thereby increasing their relative probabilities compared to states not in the support set, $\mathbf{v} \in \Omega \backslash \mathcal{D}$. Instead, the model term decreases the weights/probability corresponding to highly-probable model states, or equivalently, minimizes the partition function [GBCB16]. CD does this poorly and often diverges, while mode-assisted training achieves this with better stability and faster convergence (see Fig 4.2). We provide here an intuitive explanation of this phenomenon, while a formal treatment on this topic will be provided in the Supplementary Note 2.

The pre-training routine can be broken down in two phases. In the first phase, the training procedure attempts to discover the support $\mathcal{D}$ of the data distribution $q(\mathbf{v})$. We call this phase the "discovery phase". To better see this, consider a randomly initialized RBM with small weights. These small and uncorrelated weights give rise to RBM energies close to zero for all nodal states, or $E(\mathbf{v},\mathbf{h}) \approx 0$ for all $\mathbf{v}$ and $\mathbf{h}$, see Eq. (5.1). This results in the model distribution $p(\mathbf{v},\mathbf{h})$ being almost uniform.

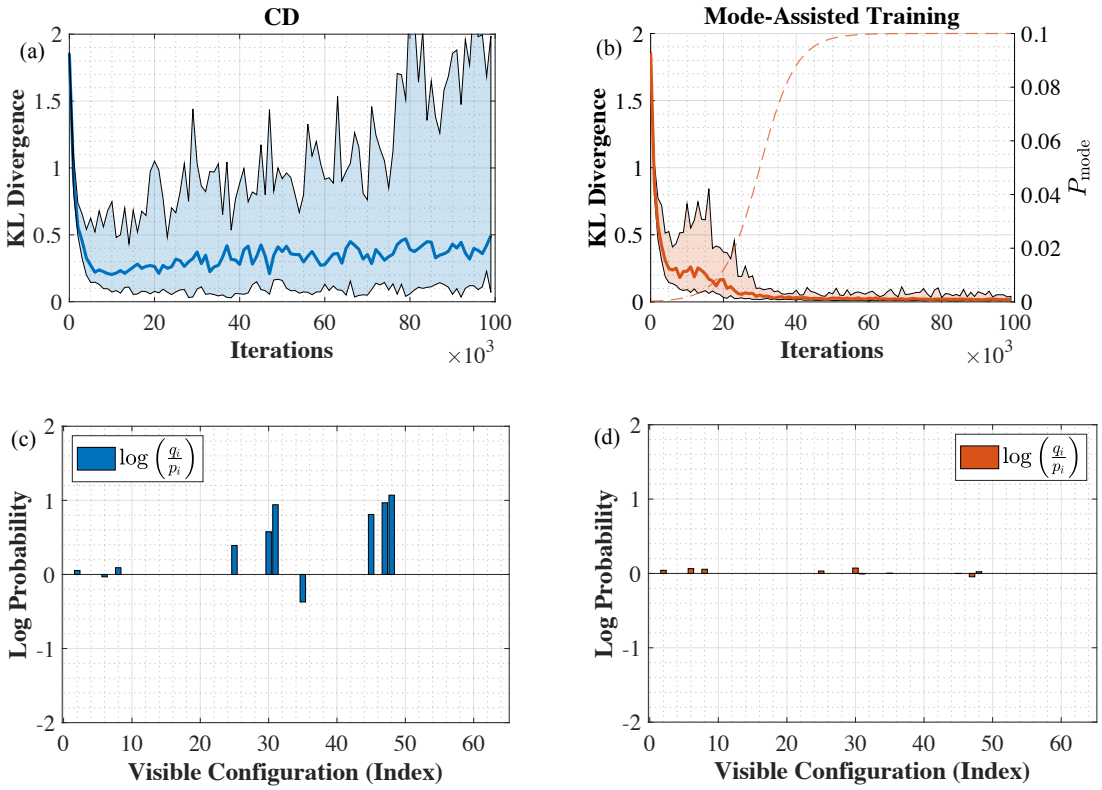


Figure 4.2: Comparison of Typical Features in Training. A training performance comparison between contrastive divergence (CD) with $k = 1$ steps of the Markov chain (in blue), and mode-assisted training (in orange) across 25 randomly generated 6×6 Restricted Boltzmann Machines with a random uniform data set of size $n_d = 10$. Panels (a) and (b) show Kullback-Leibler (KL) divergence as a function of training iterations of CD and mode-assisted training, respectively. Median KL divergence shown as the solid curves, with the shaded region defined by the maximum/minimum KL divergence at that point in training. The mode sampling probability in mode-assisted training, P_{mode}, is shown as the dotted line in panel (b). Panels (c) and (d) show the median log-differences in probability between the data (q_i) and model (p_i) distributions for CD and mode-assisted training, respectively. In both cases, the learning rate was a constant $\varepsilon^{\text{CD}} = 0.05$ for 100,000 iterations.

Therefore, we see that in the discovery phase of training, the model term plays little role in the training as it simply pushes down on the weights in a practically uniform manner, with $\langle v_i h_j \rangle_{\text{M}} \approx 0.25$. On the other hand, the data term drives the initial phase of the training by increasing the marginal probability of the visible states in the support, $\mathbf{v} \in \mathcal{D}$. We can then employ a large learning rate (say, $\varepsilon^{\text{CD}} = 1$) in the beginning of the training, driving the visible layer configurations in the dataset, $\mathcal{D}$, to high probability versus configurations outside the support.

Empirically, we find that CD training performs in the discovery phase reasonably well, and is quickly able to "find" the visible states in the support.

Now, having discovered the support, we arrive at the second phase of the training where we have to bring the model distribution as close to uniformity as possible over the support in order to minimize the KL-divergence. We call this phase the "matching phase" of the training, where we bring the model distribution as close to the data distribution as possible. CD usually performs poorly in this phase (see Fig. 4.2). To see this most directly, we simply have to consider a visible state with a slightly larger probability than the other states. It should then be necessary for the model term to locate and "push down" on this state to lower its probability, and in turn, increase the uniformity of the distribution over the support. However, for any CD approximation of the model term, this rarely happens in a timely manner as the mixing rate of the MCMC chain is far too slow to locate this state before the training diverges.

This is where samples of the mode are most effective, and can assist in the correction of the states' amplitudes. As we have anticipated, finding the modal state, $\mathbf{v}^*$, of the model distribution, $p(\mathbf{v},\mathbf{h})$ allows us to immediately locate the mode, $\mathbf{v}^+$, of the marginal probability, $p(\mathbf{v})$, and decrease the weights corresponding to that state, which in turn"pushes down" on the probability of this state through an iteration of weight updates. This "push" may result in another state "popping" up and becoming the new modal state. However, often times the probability amplitude of this new state will be less than that of the previous mode. This results in a training routine that "cycles" through any dominant state that emerges at each iteration, and the probability amplitude of the mode decreases as training proceeds until the probability amplitudes of all the states in the support become equal (see the formal demonstration of this in the Supplementary Note 2), which results in the desired uniform distribution over the support. This can be visualized as a "seesaw" between the dominant states, with the oscillation amplitude of this seesaw decaying to zero in time.

We outline the pseudo-code for mode-assisted training in Algorithm 1 and a visual

depiction of the training side by side with CD-1 on a small data set is shown in Fig. 4.2. KL divergences for CD (Fig. 2a) and mode-assisted training (Fig. 2b) are presented in the top row. In Fig. 2c, the length of the bars above or below zero imply an over- or under-estimation, respectively, of probability to a given data vector by CD. In contrast, Fig. 2d shows that the mode-assisted training algorithm searches for the state with highest probability, and decreases the weights corresponding to that state, which in turn decreases the probability assigned to it by the model, aligning it closer to the data distribution. This prevents any probability from accumulating far away from the data distribution and eventually achieves a close to perfect model. The average height of the individual divergences is exactly the KL divergence plot in Figs. 2a and 2b.

As it should now be clearer, these mode-driven updates are deviations from the gradient direction, since in general the mode over the model distribution is different from the expected value. This makes the mode-training algorithm, which mixes mode driven samples and data-driven ones, distinct from gradient descent. This is also supported by the fact that our method tends toward a particular class of distributions (uniform), when gradient descent would settle in some local minima or saddle points in the KL landscape.

Algorithm 1 Unsupervised learning of a Restricted Boltzmann Machine with the mode-assisted training algorithm

procedure MT($P_{\max}, \alpha, \beta, \{\varepsilon_n^{\mathrm{CD}}\}_{n=1}^{N}, N$)
  $\theta_0 \sim \mathcal{N}(0, 0.01)$
  for $i = 1; i \leq N; i$++ **do**
    $p_{\mathrm{mode}} \leftarrow P_{\max}\sigma(\alpha i + \beta)$
    Sample $u \sim \mathrm{U}[0,1]$
    if $u \leq p_{\mathrm{mode}}$ **then**
      $\mathbf{v}^*, \mathbf{h}^*, E_0 \leftarrow \mathrm{argmin} E(\mathbf{v}, \mathbf{h})$
      $\gamma \leftarrow \frac{-E_0}{(n+1)(m+1)}$
      $\theta_i \leftarrow \theta_{i-1} + \gamma\varepsilon_i^{\mathrm{CD}}\Delta\theta^{\mathrm{mode}}$ ▷ Eq. 4.9
    else
      $\theta_i \leftarrow \theta_{i-1} + \varepsilon_i^{\mathrm{CD}}\Delta\theta^{\mathrm{CD}}$ ▷ Eq. 4.8
    end if
  end for
  return θ_N
end procedure

The free parameters in this method are the schedules of the mode sample using $P_{\text{mode}}(n)$ (defined by $P_{\max}$, α and β in Eq. (4.7)) and the CD learning rate, ε^{CD}. With ε^{CD} fixed, we set $\varepsilon^{\text{mode}} = \gamma\varepsilon^{\text{CD}}$, where $\gamma = -E_0/[(n+1)(m+1)]$, with $E_0(<0)$ being the ground state of the corresponding RBM with nodal values $\{-1,1\}^{n+m}$. This particular choice of γ is an upper bound to the learning rate which minimizes the RBM energy variance over all states (see the Supplementary Note 2 for the proof of this statement).

We find that the mode-assisted training method is not very sensitive to the parameters chosen. In fact, as long as the mode samples are incorporated after the joint and marginal mode equilibration, the training is stabilized and the learned distribution will tend to uniformity (see also the Supplementary Note 2). This result reinforces the intuitive notion that the pushes on the mode provide a stabilizing quality to the training over CD (or any other MCMC approach), which can otherwise diverge when mixing rates grow too large at later times during training.

To realize this method in practice, one must supplement standard gradient updates with updates constructed from samples of the ground state of the RBM. Finding this ground state is equivalent to a Quadratic Unconstrained Binary Optimization (QUBO) problem, known to be nondeterministic polynomial time (NP)-hard [AB09]. Therefore, although we can compute the ground state of RBMs exactly for small datasets, for efficient mode sampling in realistically sized cases, we employ a memcomputing solver that compares favorably to other state-of-the-art optimizers in efficiently sampling the ground states of similar non-convex landscapes [TCSDV18, STDV18]. Utilizing this approach and accounting for CPU time, we find that mode-assisted training leads to models with lower KL divergences than CD. The details of our implementation, including computational complexity and energy comparisons with MCMC, can be found in Supplementary Note 3. However, in principle, one could use other optimizers for mode-assisted training.

4.2.5 Importance of Representability

Note that since mode-assisted training is driven to distributions of a particular form, instead of local minima as in the case of CD or other gradient approaches, the representability of the RBM becomes important. The ability of a RBM to represent a given data distribution is given by the amount of hidden nodes, where one is guaranteed universal representability with $n_d + 1$ hidden nodes [LRB08]. In other words, one more hidden node than the number of visible configurations with non-zero probability is sufficient (but perhaps not necessary) for general representability. In practice, this bound is found to be very conservative and typically much fewer nodes are needed for a reasonable solution.

Representability can become an issue in mode-assisted training when the parameter space of the RBM does not include the uniform distribution over the support (or a reasonable approximation). Since mode-assisted training is generally in a non-gradient direction, this means that it may settle to a worse solution than a local optimum obtainable by CD. This is a signal that more hidden nodes are required for an optimal solution.

Since most natural datasets live on a very small dimensional manifold of the total visible phase space, $|n_d|/|\Omega| \ll 1$, the amount of hidden nodes required typically scales polynomially with the size of the problem, versus the exponential scaling of the visible phase space. This makes representability not an insurmountable problem for mode-assisted training, even in full size problems. In the Supplementary Note 2, we demonstrate that regardless of representability, a mode-assisted weight update reduces the variance of the energies across the RBM states. To this end, the examples of Fig. 4.2 and Fig. 4.3 show that mode-assisted training does not necessarily fail if the number of hidden nodes is less than that needed to guarantee representability.

4.2.6 Examples of Mode-Assisted Training on Datasets

As examples of our method, we have computed the log-likelihoods achieved with mode-assisted training across two synthetic and one realistic (MNIST) datasets, and compared the results against the best achieved log-likelihoods with CD-1, PCD-1 and PT on standard RBMs, E-RBMs, and C-RBMs [MFW16]. For the small synthetic datasets we could also compute the exact log-likelihoods, thus providing an even stronger comparison. An implementation of the mode-assisted training algorithm on these synthetic data sets is available in the Supplementary Code that accompanies this paper. For the larger MNIST case, mode sampling was done via simulation of a digital memcomputing machine based on Ref. [BSDV19]. The specific details of our implementation can be found in Supplementary Note 3.

We plot an example of training progress in a moderately large synthetic problem in Fig. 4.3. Reported is the KL divergence (which differs from the log-likelihood by a constant factor independent of the RBM parameters [GBCB16]) of a slightly bigger 14×10 RBM as a function of number of parameter updates on a $L = 14$, $B = 7$ shifting bar set, for both CD-1 and mode-assisted training. We consider two learning rate schedules, constant ($\varepsilon^{\mathrm{CD}} = 0.05$) and exponential decay ($\varepsilon^{\mathrm{CD}}(n) = e^{-cn}, c = 4, n \in [0,1]$, the fraction of completed training iterations). Additionally, every time a mode sample is taken, CD is allowed to run with $k = 720$, a number scaled to the equivalent computational cost of taking a mode sample. Thus, the x-axes in Fig. 4.3 could be interpreted as wall time. The details of the computational comparison between a mode sample using memcomputing and iterations of CD are discussed in greater detail in Supplementary Note 3. In both cases, even when computational cost is factored in, mode-assisted training leads to better solutions and proceeds in a much more stable way across runs (lower KL variance at convergence). Importantly, mode-assisted training never diverges while CD oftentimes does. Following our intuition about mode-assisted training established in the "Mode Correspondence of Joint and Marginal Distributions" section, using larger learning rates in the CD-dominated phase accelerates the convergence of mode-assisted training.

It is known that using CD to train RBMs can result in poor models for the data distribution [Hin10], for which PCD and PT are recommended. We note that for the mode-assisted training employed in this paper, CD-1 was used as the gradient approximation (except in the case for MNIST where PCD-1 was used). Impressively, in all cases tested, the mode samples were able to stabilize the CD algorithm sufficiently to overcome the other, more involved approximations (PT) and model enhancements (centering).

In addition, it is clear that mode-assisted training exhibits several desirable properties over CD (or other gradient approaches). Most significantly, it seems to perform better with larger learning rates during the gradient dominated phase, and smaller learning rates when using mode samples. CD and other gradient methods generally perform better with smaller learning rates, as their approximation to the exact gradient gets better. Irrespective, even in this regime, mode-assisted training eventually drives the system to the uniform solution compared to the local optimum of CD. The main advantage is that with mode-assisted training, one can (and often should) use larger learning rates, resulting in fewer required iterations.

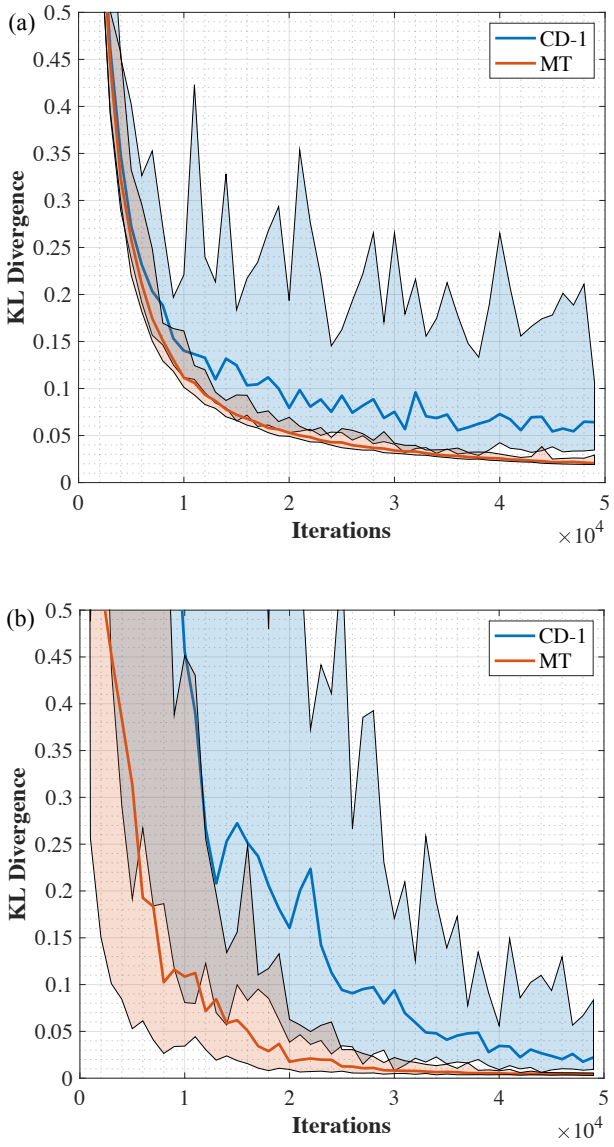


Figure 4.3: Mode-Assisted Training and Contrastive Divergence Performance on the Shifting Bar Dataset. Shown are the Kullback-Liebler (KL) divergences achieved on the binary shifting bar dataset across 25 randomly initialized 14×10 Restricted Boltzmann Machines for both Contrastive Divergence (CD) with $k = 1$ iteration and mode-assisted training (MT). In addition, every time a mode sample is taken, CD is allowed to run with $k = 720$ iterations, a number scaled to the equivalent computational cost of taking a mode sample. Thus, the x-axis can also be read as wall time. The bold line represents the median KL divergence across the runs, and the max/min KL divergences achieved at that training iteration define the shaded area. Panel (a) is with a small CD learning rate, $\varepsilon^{CD} = 0.05$. Panel (b) is with an exponentially decaying $\varepsilon^{\mathrm{CD}}(n) = e^{-cn}$ with decay constant $c = 4$ and $n \in [0,1]$ being the fraction of completed training iterations.

Table 4.1: Log-likelihood Comparisons Across Data Sets. We report the highest achieved log-likelihoods over 50,000 gradient updates on a 9×4 Restricted Boltzmann Machine (RBM) across various RBM types (standard, Enhanced-RBM, Centered-RBM) and training techniques (Contrastive Divergence (CD), Persistent-CD (PCD), Parallel Tempering (PT)) as reported in previous work [MFW16] compared with mode-assisted training (MT) on a standard RBM. In the table, rows correspond to different training techniques and columns are different data sets. For each technique, the best achieved log-likelihood score across 25 runs is reported. In parenthesis are results for a 9×9 RBM. For these small datasets we can also compare with the exact result. For the MNIST dataset, the trained networks trained had 16 hidden nodes and PCD-1 was used as the gradient update, and average log-likelihood is reported.

	S. Bar	Inv. S. Bar	Bars & Stripes	MNIST
CD-1	-20.42	-20.73	-61.08	-152.42
PCD-1	-21.71	-21.64	-57.01	-140.43
PT	-20.57	-20.57	-51.99	-142.00
MT	**-19.85**	**-19.86**	**-50.79**(**-41.82**)	-**136.42**
Exact	-19.77	-19.77	-41.59	–

For further comparison, we report in Table 4.1 results for the shifting and inverted bar, bars and stripes, and MNIST datasets obtained with mode-assisted training and those reported in past work [MFW16]. The results show mode-assisted training with a standard RBM always converges to models with log-likelihoods higher than E-RBMs, and C-RBMs trained with CD-1, PCD-1, or PT. Furthermore, the mode-assisted training log-likelihood increases with an increasing number of hidden nodes (better representability). Empirically, we also find the incredible result that with sufficient representability and the proper learning rate, mode-assisted training can find solutions arbitrarily close to the exact distribution.

4.3 Discussion

In this paper we have introduced an unsupervised training method that stabilizes gradient based approaches by utilizing samples of the ground state of the RBM, and is empirically seen to get as close as desired to a given uniform data distribution. It relies on the realization that as training proceeds, the RBM becomes increasingly frustrated, leading to the modes of the visible

layer distribution and joint model distribution becoming effectively equal. As a consequence, by using the mode (or ground state) of the RBM during training, our approach is able to "flatten" all modes in the model distribution that do not correspond to modes in the data distribution, reaching a steady state only when all modes are of equal magnitude. In this sense, the ground state of the RBM can be thought of as 'supervising' the gradient approximation during training, preventing any pathological evolution of the model distribution.

Our results are valid if the representability of the RBM is enough to include good approximations of the data distribution. Once the representability is sufficient, a properly annealed learning-rate schedule will take the KL divergence as low as desired. Increasing the number of hidden nodes increases the non-convexity of the KL-divergence landscape, easily trapping standard algorithms in sub-optimal states. In practice, after some point, increasing the number of hidden nodes will not decrease the KL divergence that a pre-training procedure actually converges to, as the trade-off between effective gradient update and representation quality is reached. We here claim that this point of tradeoff for our mode-assisted procedure is reached at far greater number of nodes than standard procedures, thus allowing us to find representations with far smaller KL-divergence. The mode-assisted training we suggest then provides an extremely powerful tool for unsupervised learning, which *i)* prevents a divergence of the model, *ii)* promotes a more stable learning, and *iii)* for data distributions uniform across some support, can lead to solutions with arbitrary high quality.

Superficially, this method resembles 'mode hopping' MCMC proposed in recent literature [SW07, LSS14], where local maxima are either found with some optimization method or assumed to be known before hand (via a dataset). However, a crucial difference between our mode-assisted training for RBMs and mode hopping algorithms is that we do not use the modal configuration to initiate a new MCMC update to improve the mixing rate. Instead, the mode itself is used to inform the weight updates directly. The difference is substantial. In fact, since higher energy states are exponentially suppressed, exposing the Markov chain to the mode will

most likely get it stuck there, which requires ad hoc constructions to recover detailed balance. Our mode-training method does not suffer from these drawbacks and is thus a more computationally efficient way to utilize the mode to train RBMs. Furthermore, we show that under a sufficiently large learning rate, sampling the global mode alone is capable of exploring efficiently a multi-modal energy landscape.

To scale our approach, one would need an efficient way to sample low-energy configurations of the RBM, a difficult optimization problem on its own. This is equivalent to a weighted MAX-SAT problem, for which there are several industrial-scale solvers available. Also, the recent successes of memcomputing on these kind of energy landscapes in large cases (million of variables) are fodder for optimism [TCSDV18, STDV18].

Finally, fitting general discrete distributions (with modes of different height) with this technique seems also within reach. In this respect, we can point to our results on the bars and stripes dataset (a non-uniform $q(\mathbf{v})$) for inspiration. We have found the best log-likelihood on that set with mode-assisted training with a lower frequency of the mode sampling, $P_{\max} = 0.1 \to 0.05$, compared to the shifting bar (a uniform set). This suggests that a general update, which properly weighs the mode sample in combination with the dataset samples, may extend this technique to general non-uniform probabilities, with the weight analogous to a tunable demand for uniformity.

Our method is useful from a number of perspectives. First, from the unsupervised learning point of view, it opens the door to the training of RBMs with unprecedented accuracy in a novel, non-gradient approach. Second, many unsupervised models are used as 'feature learners' in a downstream supervised training task (e.g., classification), where the unsupervised learning is referred to as pre-training. We suspect that starting backpropagation from an initial condition obtained through mode-assisted training would be highly advantageous. Third, the mode-assisted training we suggest can be done on models with any kind of pairwise connectivity, which include deep, convolutional, and fully-connected Boltzmann machines. We leave the analysis of these types of networks for future work.

4.4 Methods

For synthetic data, we use the commonly employed binary shifting bar and bars and stripes datasets [Mac03]. The former is defined by two parameters: the total length of the vector, L, and the amount of consecutive elements (with periodic boundary conditions), $B < L$, set to one, with the rest set to zero. This results in L unique elements in the dataset with uniform probability, giving a maximum likelihood of $L\log(1/L)$. The inverted shifting bar set is obtained by swapping ones and zeros. The bars and stripes dataset is constructed by setting each row of a $D \times D$ binary pattern to one with probability 1/2, and then rotating the resulting pattern 90° with probability 1/2. This produces 2^{D+1} elements, with the all-zero and all-one patterns being twice as likely as the others.

For a direct comparison to previous work, we followed the same training setup. [MFW16]. For the data in Table 1, a 9×4 RBM was tested on a shifting bar dataset with $L = 9$, $B = 1$ and a $D = 3$ bars and stripes dataset. Both synthetic sets were trained for 50,000 parameter updates, with no mini-batching, and a constant $\varepsilon^{\text{CD}} = 0.2$. For the MNIST dataset, a 784×16 sized model was trained for 100 epochs, with batch sizes of 100. The mode samples in both cases are slowly incorporated into training in a probabilistic way following Eq. (4.7), initially with $P_{\text{mode}} = 0$ and driven to $P_{\text{max}} = 0.1$ for the shifting bar and MNIST datasets, and $P_{\text{max}} = 0.05$ for the bars and stripes dataset. In both cases, we chose $\alpha = 20/N$ and $\beta = -6$, where N is the total number of parameter updates.

Data availability

The MNIST data set used in this paper is publicly available (`http://yann.lecun.com/exdb/mnist/`). All other data that support the plots within this paper are available from the correspond-

ing author upon request.

Code availability

The Matlab code for the training examples used in this work is provided in the Supplementary Code file which accompanies the paper.

Acknowledgments

Work supported by DARPA under grant No. HR00111990069. H.M. acknowledges support from a DoD-SMART fellowship. M.D. and Y.R.P. acknowledge partial support from the Center for Memory and Recording Research at the University of California, San Diego. All memcomputing simulations reported in this paper have been done on a single core of an AMD EPYC server.

Author contributions

M.D. has supervised and suggested the project. H.M. and M.D. conceived the idea. Y.R.P. established the theoretical framework for analyzing the algorithm. H.M. has done all the simulations reported. S.R.B. Bearden has designed the digital memcomputing machine employed in this work. All authors have discussed the results and contributed to the writing of the paper.

Competing Interests

M.D. is the co-founder of MemComputing, Inc. (`https://memcpu.com/`) that is attempting to commercialize the memcomputing technology. All other authors declare no competing interests.

Chapter 4, in full, is a reprint of the material as it appears in Manukian, Haik, Yan Ru Pei, Sean RB Bearden, and Massimiliano Di Ventra. "Mode-assisted unsupervised learning of restricted Boltzmann machines." Communications Physics 3, no. 1 (2020): 1-8. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Mode-Assisted Joint Training of Deep Boltzmann Machines

The following chapter has been submitted for publication and is available on arXiv as: Manukian, Haik, and Massimiliano Di Ventra. "Mode-Assisted Joint Training of Deep Boltzmann Machines." arXiv preprint arXiv:2102.08562 (2021).

Abstract

The deep extension of the restricted Boltzmann machine (RBM), known as the deep Boltzmann machine (DBM), is an expressive family of machine learning models which can serve as compact representations of complex probability distributions. However, *jointly* training DBMs in the *unsupervised* setting has proven to be a formidable task. A recent technique we have proposed, called mode-assisted training, has shown great success in improving the unsupervised training of RBMs. Here, we show that the performance gains of the mode-assisted training are even more dramatic for DBMs. In fact, DBMs jointly trained with the mode-assisted algorithm can represent the same data set with *orders of magnitude* lower number of total parameters compared to state-of-the-art training procedures and even with respect to RBMs, provided a *fan-in* network topology is also introduced. This substantial saving in number of parameters makes this training

method very appealing also for hardware implementations.

5.1 Introduction

One of the most highly influential models in Artificial Intelligence (AI) and deep learning in particular is the Boltzmann machine (BM). They were constructed[AHS85, Smo86] as a powerful stochastic generalization of Hopfield networks [Hop82] that possess a simple expression for their log-likelihood gradient. However, they remained impractical to train due to their reliance on high-dimensional sampling to calculate that gradient. A relatively efficient learning algorithm, called contrastive divergence (CD), was discovered for BMs with a simplified topology called restricted Boltzmann machines (RBMs)[Hin02], which have since gone on to see success in various domains [GBCB16]. However, the extension of RBMs to deep Boltzmann machines (DBMs) has been difficult [GCB12], and as such, DBMs are now mostly overshadowed by their deep feedforward cousins [GPAM$^+$14] in generative applications.

This is not due to DBM's lack of ability, but rather the absence of effective means to train these models. There remain quite a few reasons to search for better learning algorithms for DBMs, as they are a versatile computational medium. A principle use is as compact generative models for complex probability distributions in unsupervised settings, considered a critical component of the forthcoming "third-wave" of AI [Lau17]. Trained DBMs can also serve as an informed prior for feedforward networks, leading to better generalization in supervised tasks [EBC$^+$10]. In the physical sciences, DBMs serve as powerful variational representations of many body wavefunctions, more efficiently [GD17] than RBMs [CT17, MCCC19], and have potential applications in condensed matter physics and quantum computing [CCC$^+$19].

Various attempts have been made to climb the summit of DBM training [SH09, SL10, HS12, GCB13, MFW16]. Most approaches rely on pre-training by breaking up layers into RBMs and training them sequentially, after which the DBM is fine-tuned jointly. Correlations

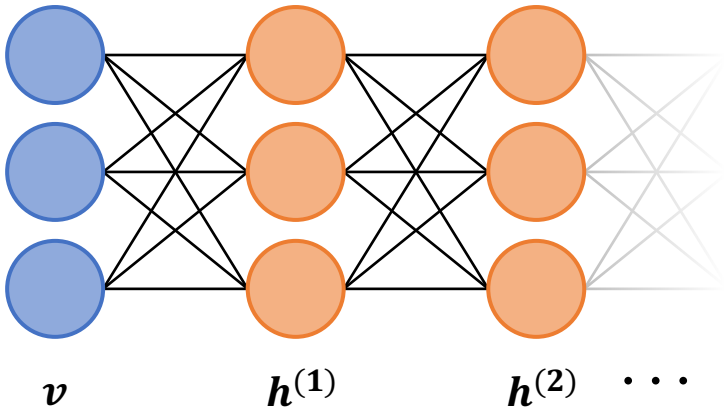


Figure 5.1: Schematic of a deep Boltzmann Machine with a visible layer, v, and hidden layers, $h^{(j)}$, with $j = 1, 2, \ldots$. Connections between nodes are symmetric and undirected, in contrast to typical directed feedforward networks.

between layers are ignored during pre-training, minimizing the potential advantages of the deep architecture and can be disruptive to joint training [MFW16]. Recently, the authors introduced mode-assisted training [MTD18, MPBDV20], which combines CD with samples of the model distribution mode. This stabilizes training, allows the learning of very accurate densities, and strikes a better tradeoff between accuracy and computational cost compared to CD. As the method is agnostic to the connectivity of the network, we can apply mode-assistance to DBMs with the hope of capturing the model capacity missed by other approaches.

In this work, we find that the benefits of mode-assisted training are even more dramatic in the case of DBMs. In fact, it produces more accurate models without requiring pre-training while also utilizing *orders of magnitude* less parameters, compared to pre-trained [SH09] or centered DBMs [MFW16]. The role of the network topology is also discussed, where we discover that DBMs are easier to train if the size of the layers decreases with depth. We evaluate the density modeling performance of mode-assisted DBMs by computing exact log-likelihoods achieved on small data sets and approximating the likelihood on the MNIST data set. The approach we propose can be extended to other types of neural networks, and is relevant also for the hardware implementation of these models, where a much smaller number of parameters directly translates into components and energy savings.

To see how mode-assisted training can be extended to deep architectures, we give a quick overview of DBMs and the basic approach to their training. DBMs are undirected weighted graphs that differentiate between n_v visible nodes, and ℓ layers of n_ℓ latent, or 'hidden', nodes, not directly constrained by the data [SH09]. We assume that $\ell > 1$ as $\ell = 1$ recovers the RBM, and like an RBM, there are no connections within a layer. Each state of the machine corresponds to an energy of the form

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(\ell)}) = &-\mathbf{a}^T \mathbf{v} - \mathbf{v}^T \mathbf{W}^{(1)} \mathbf{h}^{(1)} \\ &- \sum_{i=1}^{\ell} \mathbf{b}^{(i)T} \mathbf{h}^{(i)} - \sum_{i=2}^{\ell} \mathbf{h}^{(i-1)T} \mathbf{W}^{(i-1)} \mathbf{h}^{(i)}, \end{aligned} \tag{5.1}$$

where the biases $\mathbf{a} \in \mathbb{R}^{n_0}$, $\mathbf{b}^\ell \in \mathbb{R}^{n_\ell}$, and weights $\mathbf{W}^\ell \in \mathbb{R}^{n_{\ell-1} \times n_\ell}$ are the learnable parameters. The energy function in Eq. (5.1) induces a Boltzmann-Gibbs distribution over states,

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{\mathcal{Z}}, \tag{5.2}$$

where $\mathbf{x} = (\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(\ell)})$. The partition function, $\mathcal{Z} = \sum_{\{\mathbf{x}\}} e^{-E(\mathbf{x})}$, involves the sum of an exponentially growing number of states, making the exact computation of its value infeasible for most data sets.

During learning, a DBM is tasked to match its marginal distribution over the visible layer, $p(\mathbf{v}) = \sum_{\{\mathbf{h}\}} p(\mathbf{v}, \mathbf{h})$, to an unknown data distribution, $q(\mathbf{v})$, represented by a data set, $\mathcal{D}$. Training a DBM amounts to a search for the appropriate weights and biases that will minimize the quantity known as the Kullback-Leibler (KL) divergence between the two distributions,

$$\mathrm{KL}(q||p) = \sum_{\{\mathbf{v}\}} q(\mathbf{v}) \log \frac{q(\mathbf{v})}{p(\mathbf{v})}, \tag{5.3}$$

or, equivalently, maximizing the log-likelihood of the dataset, $\mathrm{LL}(p) = \sum_{\mathbf{v} \in \mathcal{D}} \log p(\mathbf{v})$. The optimization of the non-linear, and typically high dimensional Eq. (5.3) (or log-likelihood), is

often done via stochastic gradient descent with respect to the DBM parameters, which leads to weight updates of the form [FI12],

$$\Delta w_{ij} \propto \langle x_i x_j \rangle_D - \langle x_i x_j \rangle_M. \tag{5.4}$$

For every gradient update in Eq. (5.4), nodal statistics must be computed under two different distributions. The first one on the RHS of Eq. (5.4) is called the "data term", and is an expectation over the data induced distribution, $q(\mathbf{v})p(\mathbf{h}|\mathbf{v})$, with the network's visible layer fixed to the data. The second term on the RHS of Eq. (5.4) is called the "model term" which is an expectation over the entire model distribution in Eq. (5.2). In the case of RBMs, the data term can be sampled from exactly, but the model term must be approximated. With DBMs, the data term must also be approximated, most popularly with an iterative mean field procedure (see Methods).

In both cases, model statistics are collected via a Markov Chain Monte Carlo (MCMC) procedure dubbed 'contrastive divergence' (CD) [Hin02]. CD-k is a form of Gibbs sampling that initializes chains of length k from elements of the dataset. Trouble arises when the model distribution contains 'spurious' modes where the data distribution has negligible probability. In these cases, ergodicity breaks down, and mixing times become prohibitively long, frequently resulting in CD becoming biased enough to cause training to diverge [FI10]. Training a DBM *jointly* with CD has proven to be a formidable task. Even a two-layer DBM on MNIST has not seen success without some kind of modification. [SH09, MFW16, GCB12, DCB12]

Here, instead, we use mode-assisted training in the *joint* and *unsupervised* learning of DBMs. The essence of mode-assisted training is the replacement of some gradient updates in Eq. 5.4 with ones of the form,

$$\Delta w_{ij} \propto [x_i x_j]_D - [x_i x_j]_M \tag{5.5}$$

Where the notation $[f(\mathbf{x})]_q$ represents $f(\mathbf{x}_{\text{mode}})$, evaluated at $\mathbf{x}_{\text{mode}}$, the mode of some

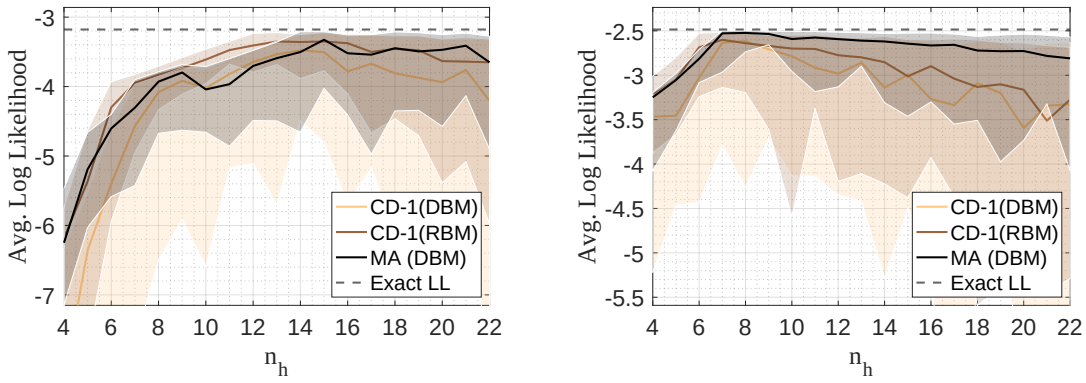


Figure 5.2: Average converged log-likelihood performance (lower bound) between RBMs trained with CD-1, mode-assisted DBMs (MA), and unassisted DBMs with CD-1. The DBMs have two hidden layers. The networks were trained on the shifting bar data set with $n_v = 24$ (left plot) and $n_v = 12$ (right plot) for 200,000 and 100,000 gradient updates respectively, following a linearly decaying learning rate schedule from $\varepsilon = 1 \to 0.001$. For the DBMs the hidden layer ratio was fixed at $\alpha = n_{h^{(2)}}/n_{h^{(1)}} = 0.2$. Performance is shown as a function of total number of hidden nodes, $n_h = n_{h^{(1)}} + n_{h^{(2)}}$. The solid lines are the median obtained across an ensemble of 50 networks, and the shaded regions enclose the 95th and 5th percentiles.

distribution, $q(\mathbf{x})$. One may employ any optimization solver to sample the mode of the above distributions. Due to its proven efficiency, here we employ a memcomputing one as reported in our previous work [MPBDV20].

The mode-assisted update can be thought of as a saddle-point approximation of an expectation [BO13]. This technique, also known as Laplace's method, is commonly employed to approximate integrals (expectations) of the form,

$$\langle f(\mathbf{x})\rangle = \frac{\int e^{-E(\mathbf{x})} f(\mathbf{x}) d\mathbf{x}}{\int e^{-E(\mathbf{x})} d\mathbf{x}} \approx f(\mathbf{x}_{\text{mode}}). \tag{5.6}$$

The weight updates driven by the mode are incorporated in a probabilistic way, with the probability of a mode driven update following a sigmoid, starting low in the initial phases of the training and reaching a maximal value at the end of training:

$$P_{\text{mode}}(n) = P_{\max}\sigma(\alpha n + \beta). \tag{5.7}$$

Here, n is the current epoch, and α, β, P_{max} control the shape of the sigmoid. Throughout the work we set, $\alpha = 20/N$ (N is the total number of epochs), $\beta = -6$ and $P_{\text{max}} = 0.1$. This schedule was introduced in Ref. [MPBDV20], based on the empirical observation that the mode samples work best when the support has been 'discovered' by CD.

The key insight of mode assisted training is *not* to approximate the expectation, but rather directly prevent spurious modes from appearing in the model distribution. This allows the inexact but efficient approximation of CD-k to minimize the KL divergence (or maximize the log-likelihood) without diverging. Searching for the mode with a highly specialized optimizer results in a better trade-off between log-likelihood performance and computation time.

5.2 Results

To illuminate the effectiveness of mode assistance on DBMs, we first evaluate performance on two differently sized synthetic shifting bar data sets. In Fig. 5.2, the mode-assisted algorithm on a two-layer DBM is compared to two baselines, CD on the same DBM and to CD on an RBM with the same number of hidden nodes. The converged log-likelihoods are reported, and results are shown as a function of the total number of hidden nodes. Overall, mode-assisted training almost always converges to more accurate densities than CD alone on a DBM, and in most cases also does better than the corresponding RBM with CD. Mode assistance is also seen to prevent the divergence sometimes seen with Gibbs sampling, as well as reducing the variance of the converged models, resulting in smaller error bars. Past a certain point, all methods expectedly incur a loss in performance as the number of hidden nodes increases for a fixed number of training iterations. Mode-assisted training suffers the least in this regard.

Although the DBMs in Fig. 5.2 possess the same total number of hidden nodes as the RBMs, they contain fewer total parameters. This means that a better trained DBM takes advantage of abstract features composition afforded by the depth of the network, mirroring similar gains

found in deep feedforward neural networks compared to single layer perceptrons [GBCB16].

Note, however, that this parameter efficiency in DBMs is *not* present when training jointly with CD. In fact, they perform systematically worse than an equivalently sized RBM in terms of log-likelihood. Mode assisted DBMs on the other hand perform as well as RBMs or better, all the while maintaining parameter efficiency.

To quantify this efficiency gain, let us consider the case of a DBM with two hidden layers. If the total number of hidden nodes is fixed, $n_h = n_{h^{(1)}} + n_{h^{(2)}}$, then a measure of parameter efficiency compared to an RBM with the same number of nodes is captured by the parameter $\alpha = n_{h^{(2)}}/n_{h^{(1)}}$. The total number of parameters (weights and biases) in an RBM with n_v visible nodes and n_h hidden nodes is,

$$n_{\text{RBM}} = n_v n_h + n_v + n_h.$$

A DBM with the same total number of hidden nodes would have the following number of parameters

$$n_{\text{DBM}} = n_v n_{h^{(1)}} + n_{h^{(1)}} n_{h^{(2)}} + n_v + n_{h^{(1)}} + n_{h^{(2)}}.$$

Considering typical training scenarios of a large visible layer, $n_v \gg n_h \gg 1$, the fraction of total parameters used in the DBM compared to the RBM (the *parameter efficiency*) can be simplified as,

$$e = \frac{n_{\text{DBM}}}{n_{\text{RBM}}} \approx \frac{1}{1+\alpha}. \tag{5.8}$$

The smaller e, the fewer parameters a DBM possesses with respect to an RBM with the same number of nodes. We then see that a naïve interpretation of Eq. (5.8) would suggest that the DBM parameter efficiency becomes more significant with increasing α, meaning the resulting DBM has more neurons in the deeper layers (fans out). However, this assumes equal log-likelihood (performance) of the two models, which is not obvious. In fact, for DBMs we

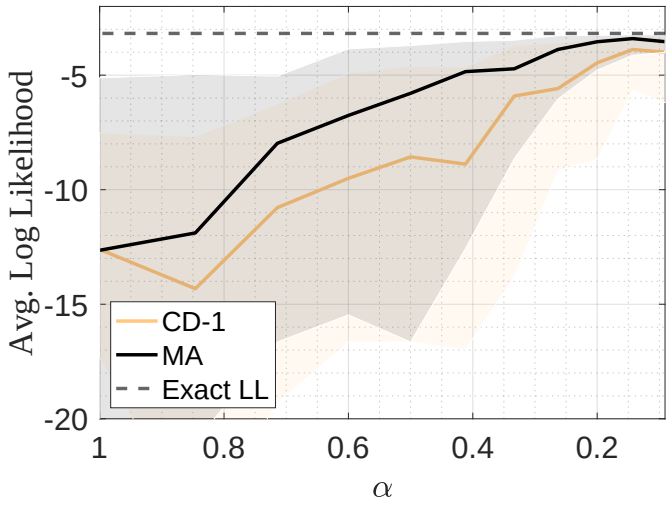


Figure 5.3: Average Log-Likelihood achieved on an $n_v = 24$ dimensional shifting bar data set as a function of DBM shape, α. The total number of hidden nodes are kept fixed at $n_{h^{(1)}} + n_{h^{(2)}} = 22$. An ensemble of 50 DBMs were trained with CD-1 and MA, where the solid line shows the median average log-likelihood achieved after 10^5 gradient iterations with a linearly decaying learning rate $\varepsilon = 1 \to 0.01$, and shaded regions delineate the 95th and 5th percentiles.

expect that a *redistribution* of the relative number of nodes in the different hidden layers plays a significant role in their performance. This tradeoff between parameter efficiency and performance merits further investigation.

This is shown in Fig. 5.3, where DBMs trained with and without mode assistance are compared as a function of α. For all cases, the mode-assisted algorithm performs better than its unassisted counterpart, which is nothing other than the confirmation of the results of Fig. 5.2. Importantly, however, we also observe that the log-likelihood performance *increases* as the network becomes less parameter efficient compared to an RBM ($\alpha \to 0$), namely when it has a *fan-in* topology. We find that a balance is reached around $\alpha \sim 0.15$, namely the second hidden layer has only about 15% of nodes than the first hidden layer. The message here is that for a *fixed* number of hidden nodes, it is best to organize the network as a fan-in DBM rather than an RBM, if one has a method to train the DBM close to capacity in the first place.

Finally, to show the substantial improvements provided by the mode-assisted training of DBMs, we compare it to the training of centered DBMs[MFW16] and pre-trained DBMs[SH09]

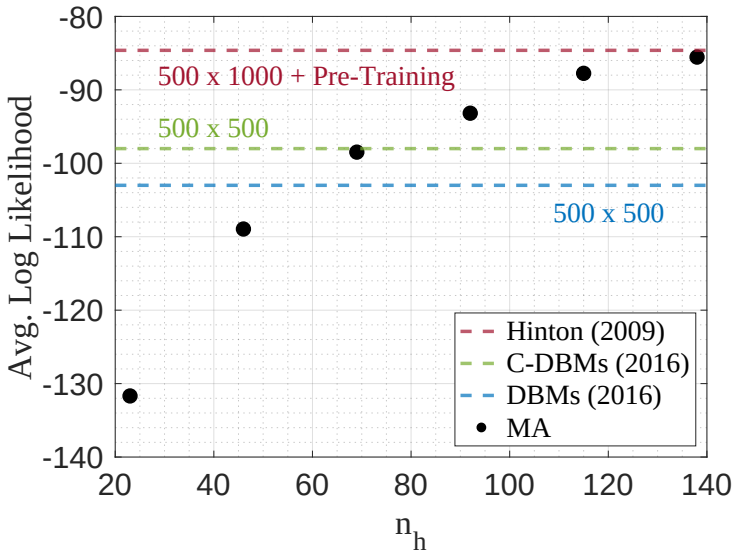


Figure 5.4: Average log-likelihood on the MNIST data set achieved after 100 epochs with mode-assisted training (MA) compared to the recent best achieved results on DBMs using CD as well as pre-training. DBMs trained with MA were all with fixed $\alpha = 0.15$ and an increasing number of hidden nodes, n_h. The learning rate was chosen to follow a linear decay, $\varepsilon = 0.05 \rightarrow 0.0005$ and no pre-training was employed. The best log-likelihood was reported out of 10 randomly initialized runs.

using CD on the MNIST data set. In Fig. 5.4, we report similar trends we observed in the smaller data sets, but scaled to more dramatic results. As the number of hidden nodes increases, small mode-assisted DBMs surpass the performance of significantly larger standard DBMs, centered DBMs and eclipse the performance of much larger pre-trained DBMs. In fact, with a DBM of only 120×18 hidden nodes (and no pre-training), trained with our mode-assisted approach, we reach the same log-likelihood of a DBM with 500×1000 hidden nodes *and* pre-training – a parameter savings of two orders of magnitude.

For a fair comparison, the total training iterations were set to 100 epochs in each case, and the log-likelihood results with our approach are shown as a function of total number of hidden nodes at a fixed ratio, $\alpha = 0.15$. The partition function was approximated using Annealed Importance Sampling (AIS) (see Methods), identical to the other referenced results [MFW16]. It's worth noting that AIS tends to *under*-approximate the partition function, leading to an over-estimate of the log-likelihood. This effect is exaggerated in larger models. Even with this negative

impact, a mode-assisted DBM with a total of 138 nodes without pre-training achieves about the same performance as one with 1,500 that was pre-trained.

5.3 Discussion

We conclude by providing an intuitive understanding of the dramatic performance improvement of the mode-assisted training over Gibbs sampling (CD). The standard method of pretraining DBMs initializes weights at a greedy starting point but critically ignores correlations within the system. The end result is a dramatic loss of parameter efficiency compared to mode-assisted joint training of DBMs. Even training methods like centering, with more advanced sampling techniques like parallel tempering [SW86], fail to train the DBM near its capacity. The issue is the reliance on Gibbs-like sampling to explore the states of the DBM, and using only these statistics to compute the likelihood gradient. The breakdown occurs when Gibbs chains fail to explore adequately the states of the system.

Because of its random-walk nature, the local dynamics of Gibbs sampling suffers long auto-correlation times when equally likely states are separated by an extensive number of variables flips. As a consequence of long mixing times between such states, statistics become heavily biased. In the initial phase of DBM training, when weights are small and randomly distributed, these issues are minimized. Interactions between nodes in this 'high temperature' regime are weak, and are well approximated by a mean-field theory [SK75]. In this phase, CD works well without much bias, and is the reason mode-assistance is really not necessary early on in the training.

However, as training continues, the DBM is effectively 'cooled' as weights learn from the data and grow larger in magnitude. Correlations between nodes become significant, and can no longer be ignored or averaged over: mean-field theory is inadequate to describe this phase. Gibbs sampling can then dramatically fail in these conditions, that is why mode-assistance is

primarily applied in this phase. During mode updates, information is propagated from the ground state, the most 'collective' or 'coordinated' state of the DBM, to all the weights. This prevents probability mass density from accumulating far away from the current state of the chain, keeping Gibbs sampling in its effective regime.

Aside from mode-assistance, we have discovered that the topology of a DBM plays a major role in its performance as well as its parameter efficiency. Too many hidden nodes act as a noise source during training, slowing down learning. Too few hidden nodes reduce the capacity of the network. For the network sizes considered, a 'sweet spot' was found where these two effects are in balance. At this stage, it is not clear if this is particular to DBMs, or there are deeper reasons why this is the case, and further work in this direction would be interesting.

Finally, mode training relies on an efficient search for the mode of a DBM, which is an NP-hard computational problem on its own. On this front, we employ a novel dynamical systems based optimization technique called memcomputing, which has shown great promise in efficient optimization of non-convex energy landscapes like the DBM [STDV19, MPBDV20]. Since the DBM energy landscape can be represented within the optimizing dynamics, a path exists to extend these systems to (learning) samplers. In doing so, the need for CD would be entirely eliminated. We leave these avenues of research to be explored in future work.

5.4 Methods

DBMs are powerful models in machine learning, but bring with them considerable computational burdens during evaluation. To deal with them, we follow the procedure outlined in Ref. [SH09]. First, due to additional complexity, the log-likelihood is not directly maximized as in the case of RBMs. Instead, a variational lower bound to the log-likelihood is optimized. Second, the most common variational form chosen leads to mean-field updates for the data term. Finally, for evaluation of log-likelihood lower bounds, an approximation of the partition function

is necessary. We provide a short description of all these approximations which are frequently encountered in DBM training scenarios.

Likelihood Lower Bound - Since the data expectation in Eq. (5.4) is no longer in closed form for the DBM, data-dependent statistics must be approximated with a sampling technique over the conditional distribution, $p(\mathbf{h}|\mathbf{v})$, where $\mathbf{h} = \{\mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(\ell)}\}$. In practice, a variational lower bound to the log-likelihood is maximized instead, which is tractable and is found to work well (as in the model term) [NH98, SH09, SH12].

The variational approximation replaces the original posterior distribution, $p(\mathbf{h}|\mathbf{v})$ with an approximate distribution, $r(\mathbf{h}|\mathbf{v})$, where the parameters of r follow the gradient of the resulting *lower bound* on the original log-likelihood [NH98],

$$\begin{aligned} \log p(\mathbf{v}) &\geq \sum_{\mathbf{h}} r(\mathbf{h}|\mathbf{v}) \log p(\mathbf{v},\mathbf{h}) + H_r(\mathbf{v}) \\ &= \log p(\mathbf{v}) - \mathrm{KL}(r(\mathbf{h}|\mathbf{v})||p(\mathbf{h}|\mathbf{v})), \end{aligned} \tag{5.9}$$

where $H_r(\mathbf{v}) = -\sum_{\mathbf{h}} r(\mathbf{h}|\mathbf{v}) \log r(\mathbf{h}|\mathbf{v})$ is the Shannon entropy of r. This variational loss simultaneously attempts to maximize the log-likelihood of the data set and minimize the KL divergence between the true conditional distribution, p and its approximation, r.

Data Term - A fully factorial mean field ansatz is often used in the variational approach, $r(\mathbf{h}|\mathbf{v}) = \prod_i r(h_i|\mathbf{v})$ with $r(h_i = 1|\mathbf{v}) = \mu_i$, with $\mu_i \in [0,1]$ randomly initialized from a uniform distribution. Maximizing the lower bound in Eq. (5.9) results in updates of the form,

$$\mu_i \leftarrow \sigma\left(\sum_j W^{(1)}_{ij} v_j + \sum_{i \neq j} J_{ij}\mu_i + b_i\right). \tag{5.10}$$

Here J is a block matrix containing the weights of the hidden nodes and b_i is the bias of the i-th hidden node. Convergence is typically fast, (in our experiments less than 30 iterations are enough)

and these states are then used to calculate the data expectation in Eq. (5.4) during the Gibbs phase of the training.

Partition Function - Computing the partition function in Eq. (5.2) exactly is infeasible for large DBMs. Its value is only required to evaluate the performance of the networks and appears in the log-likelihood as a normalizing constant. Annealed Importance Sampling (AIS) [Sal08] is the procedure often used to approximate the partition function of large RBMs and DBMs. AIS estimates the ratio of partition functions, Z_N/Z_0, using a sequence of probability distributions between a chosen initial distribution and the desired one. The initial distribution, p_0, is chosen to have an exactly known partition function and to be simple to sample from (e.g. uniform) and p_N is the desired distribution whose partition function one wants to compute.

The sequence in the case of DBMs is parameterized by β_k (inverse temperatures), giving $p_k(\mathbf{x}) = e^{-\beta_k E(\mathbf{x})}/Z_k$. Markov chains are initialized uniformly according to p_0 and $0 \leq \beta_k \leq 1$ is slowly annealed to unity according to a desired schedule, all the while allowing the chains to run. The ratio is then approximated by the product of ratios of the unnormalized intermediate distributions,

$$\frac{Z_N}{Z_0} = \frac{p_1^*(\mathbf{x})}{p_0^*(\mathbf{x})} \cdots \frac{p_k^*(\mathbf{x})}{p_{k-1}^*(\mathbf{x})}. \tag{5.11}$$

When dealing with bipartite graphs like DBMs, either all the even or all the odd layers can be analytically traced out, resulting in a tighter approximation. For a direct comparison with previous work [SH09, MFW16], we average over an identical number of 100 AIS runs with 29,000 linearly spaced intermediate distributions.

Acknowledgments

Work supported by DARPA under grant No. HR00111990069. H.M. acknowledges support from a DoD-SMART fellowship. All memcomputing simulations reported in this paper have been done on a single core of an AMD EPYC server.

Chapter 5, in full, is a reprint of the material as it appears in Manukian, Haik, and Massimiliano Di Ventra. ”Mode-Assisted Joint Training of Deep Boltzmann Machines.” arXiv preprint arXiv:2102.08562 (2021). The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion

The 'intelligence' of an artificial intelligence is the joint effort between the model used, the training algorithm employed, and the data set. Much of the current research that makes headlines aims at finding better artificial neural network models, either by scaling to larger models [BMR+20], or finding architectures that are easier to optimize (e.g. ResNet [HZRS16]). The other two aspects that influence learning receive comparatively less attention. The most popular optimization techniques are some variant of stochastic gradient descent with momentum augmentations (e.g. adaptive moment estimation, known as ADAM [KB14]). These methods have been the engine powering artificial neural networks since the 80s [RHW85]. Perhaps the biggest lesson imparted in this thesis is that the complexity of our model space is far outpacing the complexity of the underlying optimizers or learning algorithms, and leaving many important lessons behind as a result.

The Boltzmann machine has all but been abandoned in favor of other models that are more accessible to gradient based learning. There is much capacity that is left behind in these models that we have not learned how to properly train. The approach of this dissertation was to improve in artificial intelligence performance not by proposing a new model, but a new learning algorithm: namely one that itself learns from the underlying dynamics of the model. Application

of this method improved performance in every learning scenario investigated in this work.

There are many doors open to future research, probably the most obvious being applications of mode-assistance to more data sets, and larger problems. Also, while mode-assistance looks for ground states of the model, it can be extended to exploring excited states of the model as well, effectively subverting the need for CD entirely. It is my belief that these models still have much to teach us, and we should be more patient in learning where the training difficulty lies and complexifying our learning algorithms to meet these difficulties, rather than abandoning them for easier to train models. To meet these challenges, our methods of training intelligent systems must themselves become intelligent. No one is at our heels in the creation of artificial intelligence, we have the luxury of going slowly, building understanding and taking our time. Many of the difficulties are subtle and likely require slow, careful thinking. Nature will not give us the answer to these questions for free, we must earn them.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, 2009.

[AH15] Steven H Adachi and Maxwell P Henderson. Application of quantum annealing to training of deep neural networks. *arXiv preprint arXiv:1510.06356*, 2015.

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[BCMR10] Zhengbing Bian, Fabian Chudak, William G Macready, and Geordie Rose. The ising model: teaching an old problem new tricks. *D-wave systems*, pages 1–32, 2010.

[BDHS11] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM. J. Matrix Anal. & Appl.*, 34:866–901, 2011.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[Ben09] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[BMTD18] Sean R. B. Bearden, Haik Manukian, Fabio L. Traversa, and Massimiliano Di Ventra. Instantons in self-organizing logic gates. *Physical Review Applied*, 9(3), mar 2018.

[BO13] Carl M Bender and Steven A Orszag. *Advanced mathematical methods for scientists and engineers I: Asymptotic methods and perturbation theory*. Springer Science & Business Media, 2013.

[BPDV20] Sean RB Bearden, Yan Ru Pei, and Massimiliano Di Ventra. Efficient solution of boolean satisfiability problems with digital memcomputing. *Scientific reports*, 10(1):1–8, 2020.

[BSDV19] S. R. B. Bearden, F Sheldon, and M Di Ventra. Critical branching processes in digital memcomputing machines. *EPL (Europhysics Letters)*, 127(3):30005, 2019.

[BWP+17] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[CCC+19] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborova. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[CHM+15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[CK76] Leon O Chua and Sung Mo Kang. Memristive devices and systems. *Proceedings of the IEEE*, 64(2):209–223, 1976.

[CT17] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[Dat10] Bishwa Nath Datta. *Numerical Linear Algebra and Applications*. SIAM, 2010.

[DCB12] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. On training deep boltzmann machines. *arXiv preprint arXiv:1203.4416*, 2012.

[DPG+14] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[DT17] M. Di Ventra and F. L. Traversa. Absence of periodic orbits in digital memcomputing machines with solutions. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27:101101, 2017.

[DT18] Massimiliano Di Ventra and Fabio L. Traversa. Memcomputing: Leveraging memory and physics to compute efficiently. *J. Appl. Phys.*, 123:180901, 2018.

[DVP13a] M. Di Ventra and Yuriy V. Pershin. The parallel approach. *Nature Physics*, 9:200–202, April 2013.

[DVP13b] Massimiliano Di Ventra and Yuriy V. Pershin. The parallel approach. *Nature Physics*, 9:200–202, 2013.

[DVT17] M. Di Ventra and Fabio L. Traversa. Absence of chaos in digital memcomputing machines with solutions. *Phys. Lett. A*, 381:3255, 2017.

[DVT18] Massimiliano Di Ventra and Fabio L. Traversa. Perspective: Memcomputing: Leveraging memory and physics to compute efficiently. *Journal of Applied Physics*, 123(18):180901, may 2018.

[DVTO17] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. *Ann. Phys. (Berlin)*, 529:1700123, 2017.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[Fey82] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[FI10] Asja Fischer and Christian Igel. Empirical analysis of the divergence of gibbs sampling based learning algorithms for restricted boltzmann machines. In *International conference on artificial neural networks*, pages 208–217. Springer, 2010.

[FI12] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Fre93] Gottlob Frege. *Gottlob Frege: Basic laws of arithmetic*, volume 1. Oxford University Press (UK), 1893.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, Cambridge, 2016.

[GCB12] Ian J Goodfellow, Aaron Courville, and Yoshua Bengio. Scaling up spike-and-slab models for unsupervised feature learning. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1902–1914, 2012.

[GCB13] Ian J Goodfellow, Aaron Courville, and Yoshua Bengio. Joint training deep boltzmann machines for classification. *arXiv preprint arXiv:1301.3568*, 2013.

[GD17] Xun Gao and Lu-Ming Duan. Efficient representation of quantum many-body states with deep neural networks. *Nature communications*, 8(1):1–6, 2017.

[Göd31] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

[GPAM+14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[HA28] D Hilbert and W Ackerman. Theoretische logik. *Julius Springer, Berlin*, 1928.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara N Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[Hil02] David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[Hin02] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[Hin06] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Training*, 14(8), 2006.

[Hin10] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HS12] Geoffrey E Hinton and Russ R Salakhutdinov. A better way to pretrain deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2447–2455, 2012.

[HSA84] Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IEE08] IEEE Std. 754-2008, pp 1-58. *IEEE Standard for Binary Floating-Point Arithmetic*, 2008.

[ILBH+11] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Julia Hamilton, Andre van Schaik, Ralph Etienne-Cummings, Tobi Delbrunck, Shih-Chii Liu, Piotr Dudek, Philipp Hafliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:73, 2011.

[Int16] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Lau17] John Launchbury. A darpa perspective on artificial intelligence. *Retrieved November*, 11:2019, 2017.

[LB95] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[Lit74] William A Little. The existence of persistent states in the brain. *Mathematical biosciences*, 19(1-2):101–120, 1974.

[LJ09] Mantas Lukosevicius and Herbert Jaeger. Survey: Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 32(3):127–149, 2009.

[LRB08] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

[LS10] Philip M Long and Rocco A Servedio. Restricted boltzmann machines are hard to approximately evaluate or simulate. 2010.

[LSS14] Shiwei Lan, Jeffrey Streets, and Babak Shahbaba. Wormhole hamiltonian monte carlo. In *AAAI*, pages 1953–1959, 2014.

[Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, New York, 2003.

[MCCC19] Roger G Melko, Giuseppe Carleo, Juan Carrasquilla, and J Ignacio Cirac. Restricted boltzmann machines in quantum physics. *Nature Physics*, 15(9):887–892, 2019.

[MDMM11] Lei Ma, Kevin Dickson, John McAllister, and John McCanny. Qr decomposition-based matrix inversion for high performance embedded mimo receivers. *IEEE Trans. On Signal Processing*, 59:1858–1867, April 2011.

[MDV21] Haik Manukian and Massimiliano Di Ventra. Mode-assisted joint training of deep boltzmann machines. *arXiv preprint arXiv:2102.08562*, 2021.

[MFW16] Jan Melchior, Asja Fischer, and Laurenz Wiskott. How to center deep boltzmann machines. *The Journal of Machine Learning Research*, 17(1):3387–3447, 2016.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MPBDV20] Haik Manukian, Yan Ru Pei, Sean RB Bearden, and Massimiliano Di Ventra. Mode-assisted unsupervised learning of restricted boltzmann machines. *Communications Physics*, 3(1):1–8, 2020.

[MPV87] Marc Mézard, Giorgio Parisi, and Miguel Angel Virasoro. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications*, volume 9. World Scientific Publishing Company, 1987.

[MTD17] H. Manukian, F. L. Traversa, and M. Di Ventra. Memcomputing numerical inversion with self-organizing logic gates. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–6, 2017.

[MTD18] H. Manukian, F. L. Traversa, and M. Di Ventra. Accelerating deep learning with memcomputing. *arXiv:1801.00512*, 2018.

[MTDV19] Haik Manukian, Fabio L Traversa, and Massimiliano Di Ventra. Accelerating deep learning with memcomputing. *Neural Networks*, 110:1–7, 2019.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[NH98] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[OF97] Stuart F. Oberman and Michael J. Flynn. Division algorithms and implementations. *IEEE Trans. Comput.*, 46(8):833–854, August 1997.

[Par09] Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Inc., 2009.

[PMDV19] Yan Ru Pei, Haik Manukian, and Massimiliano Di Ventra. Generating weighted max-2-sat instances of tunable difficulty with frustrated loops. *arXiv preprint arXiv:1905.05334*, 2019.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[RMDB18] E. Romero Merino, F. Mazzanti Castrillejo, J. Delgado Pin, and D. Buchaca Prats. Weighted Contrastive Divergence. *ArXiv e-prints*, January 2018.

[RW10] Bertrand Russell and Alfred North Whitehead. Principia mathematica vol. i. 1910.

[Sal08] Ruslan Salakhutdinov. Learning and evaluating boltzmann machines. 2008.

[Sch15] Jurgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SCTDV19] Forrest Sheldon, Pietro Cicotti, Fabio L Traversa, and Massimiliano Di Ventra. Stress-testing memcomputing on hard combinatorial optimization problems. *IEEE transactions on neural networks and learning systems*, 31(6):2222–2226, 2019.

[SH09] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455, 2009.

[SH12] Ruslan Salakhutdinov and Geoffrey Hinton. An efficient learning procedure for deep boltzmann machines. *Neural computation*, 24(8):1967–2006, 2012.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SK75] David Sherrington and Scott Kirkpatrick. Solvable model of a spin-glass. *Physical review letters*, 35(26):1792, 1975.

[SL10] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 693–700, 2010.

[SMH07] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007.

[Smo86] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[SPMH03] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[SRPJ15] Ahmad Muqeem Sheri, Aasim Rafique, Witold Pedrycz, and Moongu Jeon. Contrastive divergence for memristor-based restricted boltzmann machine. *Engineering Applications of Artificial Intelligence*, 37:336–342, 2015.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[ST10] Ilya Sutskever and Tijmen Tieleman. On the convergence properties of contrastive divergence. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 789–795, 2010.

[STDV18] Forrest Sheldon, Fabio L Traversa, and Massimiliano Di Ventra. Taming a non-convex landscape with dynamical long-range order: memcomputing the ising spin-glass. *arXiv preprint arXiv:1810.03712*, 2018.

[STDV19] Forrest Sheldon, Fabio L Traversa, and Massimiliano Di Ventra. Taming a non-convex landscape with dynamical long-range order: Memcomputing ising benchmarks. *Physical Review E*, 100(5):053311, 2019.

[SW86] Robert H Swendsen and Jian-Sheng Wang. Replica monte carlo simulation of spin-glasses. *Physical review letters*, 57(21):2607, 1986.

[SW07] Cristian Sminchisescu and Max Welling. Generalized darting monte carlo. In *Artificial Intelligence and Statistics*, pages 516–523, 2007.

[Sze11] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2011.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[TB12] Fabio Lorenzo Traversa and Fabrizio Bonani. Improved harmonic balance implementation of floquet analysis for nonlinear circuit simulation. *Inter. J. Elec. and Comm.*, 66(5):357 – 363, 2012.

[TB13] Fabio Lorenzo Traversa and Fabrizio Bonani. Selective determination of floquet quantities for the efficient assessment of limit cycle stability and oscillator noise. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 32(2):313–317, 2013.

[TCSD18] F. L. Traversa, P. Cicotti, F. Sheldon, and M. Di Ventra. Evidence of exponential speed-up in the solution of hard optimization problems. *Complexity*, 2018:7982851, 2018.

[TCSDV18] Fabio L Traversa, Pietro Cicotti, Forrest Sheldon, and Massimiliano Di Ventra. Evidence of exponential speed-up in the solution of hard optimization problems. *Complexity*, 2018, 2018.

[TD15] Fabio Lorenzo Traversa and Massimiliano Di Ventra. Universal memcomputing machines. *IEEE Trans. Neural Netw. Learn. Syst.*, 26(11):2702, 2015.

[TDH16] J. Tang, C. Deng, and G. B. Huang. Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, 27(4):809–821, April 2016.

[TDV17a] Fabio L. Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(2):023107, 2017.

[TDV17b] Fabio L Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(2):023107, 2017.

[TH09] Tijmen Tieleman and Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1033–1040, 2009.

[TPDV13] Fabio L. Traversa, Y. V. Pershin, and M. Di Ventra. Memory models of adaptive behavior. *IEEE Trans. Neural Netw. Learn. Syst.*, 24(9):1437–1448, Sept 2013.

[TRBDV15] Fabio L. Traversa, Chiara Ramella, Fabrizio Bonani, and M. Di Ventra. Memcomputing np-complete problems in polynomial time using polynomial resources and collective states. *Science Advances*, 1:e1500031, Jul 2015.

[Tur36] Alan M. Turing. On computational numbers, with an application to the entscheidungsproblem. *Proc. of the London Math. Soc.*, 42:230–265, 1936.

[TUR50] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950.

[vN58] John von Neumann. The computer and the brain. *New Haven, Conn.: Yale Uni*, 1958.

[vN93] John von Neumann. First draft of a report on the edvac. *Annals of the History of Computing, IEEE*, 15(4):27–75, 1993.

[WCB19] Jun Wang, Gert Cauwenberghs, and Frédéric D Broccard. Neuromorphic dynamical synapses with reconfigurable voltage-gated kinetics. *IEEE Transactions on Biomedical Engineering*, 67(7):1831–1840, 2019.

[WKS14] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *arXiv preprint arXiv:1412.3489*, 2014.

[WSG94] J. H. Winters, J. Salz, and R. D. Gitlin. The impact of antenna diversity on the capacity of wireless communication systems. *IEEE Transactions on Communications*, 42(234):1740–1751, Feb 1994.