An analysis of the dichotomy between neural network theory and application

Abstract:

This is a survey of works whose aim is to showcase and analyze the divide between theoretical and practical research presently seen on neural networks. Within, selected theorems are discussed along with an analysis of the impact their results have on applications as well as possible directions for future work.

Introduction and Background:

The conception of the McCulloch Pitts (MP) network in 1943 is widely considered to be the origin of modern neural network theory. Utilizing ideas and formalisms similar to those introduced by Alan Turing in his famous 1936 paper on computable numbers [28], and as one of the first computational analogs to intelligence, it offered a presentation that could be both rigorously analyzed as well as a framework which might serve as the foundation for a deeper theory. *A Logical Calculus of the Ideas Immanent in Nervous Activity*, generated a great deal of interest in both the theory[1] as well as the possible integration of neural networks in practical problems.

The surge of interest generated by these new models was closely followed by scrutiny and criticism. Taking into account reasonable hardware limitations for their time, Marvin Minsky and Seymour Papert showed in their criticism of a version of Rosenblatt's learning model that the perceptron (a model heavily inspired by the MP network) was 'weak' computationally and that there existed a large set of functions (those which are linearly inseparable in low dimensions) which it could not represent [4]. Coupled with outside influences such as diminished funding and support for artificial intelligence, Minsky and Papert's 1969 paper is widely cited as a main contributor to the start of the first major 'AI winter'.

Holding true to the pessimistic predictions of Minsky and Papert, MP networks and their contemporaries struggled to overcome a lack of efficient training methods. While work such as Hebb's theory of learning [30] and refined forms of Rosenblatt's perceptron [3] made great strides to remedy this problem, their techniques were unscalable to larger and deeper networks. In fact, it was not until the integration of continuously differentiable activation functions and the refinement of backpropagation (around the late 1980s), that threshold circuits and perceptrons

[1] The definition of an important class of languages in computational complexity theory known as finite state machines has roots attributed to the influences of the MP neuron [29].

obtained efficient training algorithms through approximating the behavior of backpropagation [44].

More recently, catalyzed by advancements in hardware, scalable learning methods, and the curation of enormous datasets and propelled by empirical successes in previously unsolved problems [14], [33], the AI community's interest has been rekindled in deep neural networks due to their capacity for computationally tractable automated feature extraction on large datasets. In spite of the renewed attention, theory regarding the complexity of such deep architectures still lags behind. Modern theory struggles to understand how or why deep models achieve the performances that they do. The few general results regarding the computational complexity of these network models are mostly limited to shallow (2-3 layer) examples and often paint a picture of intractability with their exponential bounds on size and weights required for the computation of arbitrary functions.

Our goal is to explore the dichotomy between the theoretical and practical aspect of research on neural networks. Below, we survey some key results in the complexity analysis of various models as well as discuss their impact on the training and implementation of newer iterations.

The first section presents selected attributes by which networks may be categorized, with a focus on reductions between models. Next we cover the complexity of our baseline model, the threshold circuit, when certain parameters such as weight, size, and depth are restricted. A brief section on proposed hierarchical orderings of networks such as VC dimension, reductions to TC^0, and trajectory length follows. Finally some open problems are presented. An appendix for field specific terms with rigorous definitions are provided after the citations. In text definitions to terms are provided in a more simplified language.

Categorizing networks by attributes:

Here we make an attempt at generally categorizing neural networks by discussing properties of their architecture in isolation and presenting reductions to threshold circuits as a baseline model. Because the types of networks in literature are so numerous[2] and existing theorems so exceedingly specific in the models they affect, this is not yet possible for a majority of the categories presented. The four attributes below are the ones for which the most general theorems could be found, even still, many results are necessarily dependent on other parameters such as activation function and depth.

[2] http://www.asimovinstitute.org/neural-network-zoo/ provides an overview of the broadest categories

It is important to note that: not all theorems sampled below are constructive and may as a consequence be non-uniform. Thus, sufficient care is necessary when translating theorems proven for one architecture into another to ensure the computability of the reduced model.

Throughout this section, where available, we try to order networks by the cardinality of the set of boolean functions they may compute, making it helpful to to view neural networks as function approximators.

Activation functions

Activation functions suffer from the same problem of general categorization as network architectures: in that so many different types exist in literature. To find a meaningful category of separation we consider the fact that a large contributing factor to the first major AI winter was due to a lack of effective training methods for the architectures of the time. Backpropagation networks played a large role in revitalizing interest, and thus, the first category of separation we consider will be non-zero first-order-differentiable versus all other activation functions[3].

Boolean and threshold circuits fall under the general case. They may be naturally modeled by circuits, finite automata, and other concepts from complexity theory. In the differentiable case, we focus on sigmoid activations whose composed networks are more suited to representations from calculus and topology [59], [56].

Viewed in isolation, it is easy to see the differences between individual functions (i.e. single linear threshold and boolean gates have outputs restricted to the boolean domain while sigmoidal gates have outputs in the reals). These separations are also not difficult to show when propagated in shallow (2-3 layer) networks.

> **Theorem** *Depth 2 sigmoid networks of constant size and poly-size weights with respect to its input may be represented optimally by an equivalent depth 2 threshold circuit of exponential size [43].*

Ideally, we would like theorems which extend these single neuron and shallow network results to deeper and larger systems so that general classifications can be made. Based on the what has so far been presented, is not unreasonable to assume that differences between activation functions should become increasingly stark when propagated in deeper, polynomial sized models; empirically this has indeed been shown to be the case by all the success that continuous activations have had over their discrete counterparts. However, a result on strict general

[3] Non-differentiable functions are further separated into continuous and discrete cases which are not covered here. These two cases are also referred to as digital and analog in the case of boolean circuits in several papers [http://cognet.mit.edu/journal/10.1162/089976699300016656]

separations between the sets of computable boolean functions of deeper networks differing in activation functions seems to point in the other direction:

> **Theorem** *Polynomially sized sigmoid networks with polynomially bounded weights compute the same boolean functions as threshold circuits with the same size and weight bounds when both have depth greater than 2 [24].*

The conflicting conclusions this theorem gives versus observed empirical evidence points towards two differing possibilities: either deep threshold circuits are fundamentally different from backpropagation networks computationally or, they become equivalent after reaching a constant depth. In the second case, the lack of empirical success with threshold circuits would then have to be attributed to flaws in the training methods prescribed, preventing them from achieving their full computational potential.

These mutually exclusive scenarios provide conflicting motivations for research. If deep threshold circuits are indeed weaker than their sigmoid counterparts, then it would be optimal to focus efforts on improving learning methods solely for sigmoid networks as is the current trend. In the second case, since threshold functions are more memory efficient and faster to compute than sigmoids, it would instead be prudent to improve training methods for threshold circuits. Alternatively, it could be possible that no analogous efficient training algorithm exists for threshold circuits as it does for sigmoid networks in which case the first method is still optimal.

Other axis of comparison for activation functions have also been studied. For example, it has been shown that for activation functions f which have convergent asymptotically bounded behavior satisfying:

$$lim_{x \to inf} f(x) = t_+$$

$$lim_{x \to -inf} f(x) = t_-$$

$$t_+ \neq t_-$$

It becomes possible to construct a network of these activations of size $|S|$ which simulates any arbitrary finite automata with state space S [52]. Finite automata may be used to describe the behavior of modern computers so this result has implications for ‘trainable’ computers. However, the state space of any finite automata can be exponential with respect to its size and it has been shown that even depth 2 threshold circuits are able to compute all boolean functions when exponential size is a condition.

Weight domains

Oftentimes analysis on neural networks will assume weights to exist in the reals, however, as applications are limited to finite precision, it is useful to consider the amount of memory each weight requires for expression. The integers are useful in this regard as there is no ambiguity in how many bits they require for encoding. Results for integral weighted threshold circuits may then be generalized to rational and irrationally weighted threshold circuits. Inclusions are trivial to show between networks utilizing weights in the reals, rationals, and integers but similarly to activation functions, strictness is much harder to prove. The two following theorems provide a reduction from arbitrarily weighted threshold circuits to unweighted threshold circuits:

Theorem *Any threshold gate with* n *inputs can be represented as an integral weighted threshold gate with maximal weight* w_i *such that* $|w_i|, |T| \leq 2^{-n}(n+1)^{(n+1)/2}$

Theorem *Integral weighted threshold gates with maximal weight* $|w|$ *and fan-in polynomial in the input size* n *may be represented by a depth 2 unweighted threshold circuit of size* $O(n^a)$ *for a fixed constant* a *[35].*

As unweighted threshold gates can easily be converted to a majority gate by the introduction of 'dummy' input gates (adding either always active or always inactive input gates until the threshold condition is satisfied), a corollary to the above theorem provides a reduction from integral weighted threshold gates to majority circuits:

Theorem *Integral weighted threshold gates with maximal weight* $|w|$ *and fan-in polynomial in the input size* n *may be represented by a depth 2 majority circuit with size* $O(n^a)$ *for a fixed constant* a.

As majority and unweighted threshold gates only require one bit to represent each of their parameters, the total number of bits required to encode these networks is $O(e)$ with e being the number of edges in the network graph which is upper bounded by $O(n^a)$. Another immediate result, tells us that any depth d integral weighted threshold circuit requires at most a depth $2d$ unweighted threshold circuit with size polynomial in n to represent which may be constructed by simply replacing every integral threshold gate with its corresponding majority circuit.

Theorem *Fixed depth* d *threshold circuits with unbounded weight can be represented by depth* $2d$ *polysize weighted threshold circuits.*

Relaxing the size restrictions from polynomial to super-exponential yields a reduction which is closer in depth:

Theorem *Fixed depth d general unbound weighted threshold circuits may be computed by depth $d+1$ unweighted threshold circuits[4] in size super-exponential in d [35] (provides a non constructive proof) [51] (provides a constructive proof).*

For shallow threshold circuits, a tractable reduction to majority circuits may also be obtained. Unfortunately the tractibly computable intersection of models covered by this theorem and the theorem from the previous section (stating that polysize, depth ≥ 2 sigmoid networks compute the same boolean functions as threshold circuits with the same dimensions) is small. The super exponential size bound becomes a limiting factor at around $d > 16$. Many models perform well enough with $d \leq 16$ and so this window tells us that for boolean functions, sigmoid networks may be reduced at least in theory to reasonably sized majority circuits. Another less interesting corollary is that the minimum amount of space required for the weights of a sigmoid network computing an arbitrary boolean function will be bounded above by a value super exponential in the depth of the network.

Depth

The namesake of the modern AI field, the depth of a model appears to heavily influence the range of functions it can represent [12]. Analysis of depth's contribution to the various successes in image classification, object detection, and NLP tasks is a salient topic of research. Many ongoing works in complexity theory focuses on relating depth to computational power such as proving the existence of a depth hierarchy on the circuit classes AC^0, NC^0, and TC^0. The results provided for these circuit families have analogous results for shallow sigmoid neural networks but they become less useful as depth increases. This is in part because AC^0 and NC^0 are limited to AND, OR gates, of which a circuit of at least depth 2 is required to emulate a majority gate and TC^0 has only been separated up to depth 3 [9], [27]. Similarly to TC^0, sigmoid networks have also only been strictly separated up to depth 3. Depth 1 sigmoid networks cannot express functions requiring composition and are thus limited to monotone outputs while $d \geq 2$ may compute non-monotone functions. A more complicated proof separating depth 2 and depth 3 polysize networks with 'reasonable' activations is also available.

Theorem *There exists a function computable by a depth 3 poly-size neural network with reasonable activations which cannot be approximated by a depth 2 neural network in less than exponential size [47].*

Past depth 3, to the best of our knowledge, no more general separations have been made; no polysize reductions between networks of differing size have been made either. However when

[4] The authors use the notation LC instead of TC to represent threshold circuits

using other metrics of computation potential, sigmoid networks have been shown to be separated up to arbitrary n. We discuss this more in the section on representations of power.

Cyclicity

A network is said to be cyclic if any neuron's output depends on the outputs from neurons in the same layer and is acyclic otherwise. Threshold circuits and Restricted Boltzmann Machines (RBNs) are canonical examples of acyclic networks and recurrent neural networks (RNNs) are the most prevalent acyclic architectures.

Adding cycles makes networks much harder to analyze as their outputs can no longer be represented purely as a composition of the activations on the inputs. Cyclicity does fill a purpose however, as certain functions and problems are either ill suited to representation by feedforward architectures or may have their size and depth greatly reduced by the addition of loops either through an exact transformation or approximate compression [37], [40]. Additionally, evidence supporting the claim that many natural biological networks are in fact cyclic have also been discovered [13].

> **Theorem:** *There exists a class of functions for which the tight upper size and depth bound of an equivalent cyclic boolean circuit is smaller than the optimal acyclic architecture's size and depth by a half [50].*

The most common cyclic network type, RNNs, also have a different input style than normal threshold circuits. RNNs receive their input sequentially (over time) while threshold circuits receive inputs combinationally (all at once). Cyclicity essentially functions as an increase to the complexity to the activation of a network (each layer may add multiple function compositions to the final output) without altering depth. In the sequential case, addition of cyclicity also introduces a form of temporal memory expressed through parameters stored in each neuron (such as LSTMs). General reductions between networks differing in input styles do not exist as combinational networks must somehow receive an encoding of an arbitrarily long sequence of inputs to emulate a sequential network.

	Acyclic	Cyclic
Combinational	Threshold Circuits	Hopfield Networks
Sequential	MP Networks	RNNs

In boolean circuit theory, networks branch into even smaller subcategories separated by differing wiring (synapse) and gate computation delays and initial conditions. For the sake of simplicity, in the theorems presented below, we assume initial values are set to 0 and synapse and gate computation delays are both constant and equivalent for all synapses and gates. In the

case of boolean circuits, these assumptions ensure that the resulting model are reducible to a combinational acyclic circuit [46] and for general networks, it ensures the correctness of the reduced model.

> **Theorem** *Deterministically updated, discrete time, constant initial state cyclic networks of depth d and size s which reach a stable state within a fixed time t may be represented by an acyclic network of depth $d * t$ and size $O(s * t)$ with the same input scheme [9].*

This is achieved by 'unrolling' the cyclic network (adding a new neuron layer for the set of neurons who's afferent nodes changed in the previous timestep). As an example, recurrent layers in RNNs are generally introduced as a many layered feedforward structure with sequential input with the number of layers depending on the length of the input.

There are two major issues with this construction. The first, a family of feedforward networks will be required to emulate a cyclic network with sequential input across all input lengths. The second problem is that the construction from combinational cyclic to acyclic models depends on the halting time of the cyclic network (a problem which quickly becomes infeasible when t becomes large and undecidable in the general case as certain network models are Turing complete).

Specific models such as Hopfield nets have known halting times [16] but do not necessarily have constant synaptic delays in their implementation. A reduction between cyclic and acyclic models independent of halting time is not yet known to exist. Neither stochastically updated nor continuous updating cyclic networks appear to have a general acyclic reduction either.

Other attributes:

Many more specific attributes exist for which specific results exist to separate specific classes of networks which are not discussed at length in literature yet still separate large categories of networks.

Synchronicity[5]

GPU advancements have been seen as instrumental in enabling the training of deeper networks due to their capability to quickly perform matrix operations in parallel. Ideally, synchronicity should not be a category by which neural networks are separated as we would hope that the parallel implementation of a model does not change its behavior; while it may be

[5] [41] goes much more in depth on the topic of synchronicity in neural networks

true that the computational power of a model is not affected by introducing synchronicity, the main concern is that training accuracy may be affected.

Acyclic networks may update individual layers in parallel as the neurons will not have conflicting dependencies on neurons in the same or preceeding layers, resulting in an approximately logarithmic scale increase in both training and test speed. Results become much more interesting in the context of a cyclic architecture. The main concern regarding synchronous updates is that accuracy or convergence behaviors may be affected. Fortunately, it has been shown that synchronous computation does not affect the set of stable states of networks with a certain type of activation (a set which includes sigmoid, ReLU, linear, and thresholds) [36], [39] so only convergence behavior need be assessed in the most general cases. Of course, during implementation, race conditions must also be taken into account.

Certain cyclic networks have naturally parallelizable components such as the convolutional layer for RNNs which act much like a typical feedforward network in that synchronicity can be applied on a layer by layer basis by viewing the discrete time steps as separate layers in the network. For the components where an equivalent parallel implementation is not readily apparent, an acceptable tradeoff between accuracy and speed can usually be made [41].

Unlike neuron updates, parallel weight updates have the potential to affect training speed, accuracy, as well as convergence behavior [15]. Hopfield networks trained using the typical training scheme exhibit this behavior:

> **Theorem** *Hopfield networks converge to a stable state when operating in serial mode (asynchronously), converge to cycles of length at most 2 when in parallel mode (fully synchronous) and cycles of length 4 when in parallel mode with antisymmetric weights [16].*

Computational and representational bounds

A primary goal of introducing the reductions in the previous section was so more general results showcasing the effects different parameters have on the computational potential of threshold circuits could be presented here.

Unweighted majority circuits with V gates can be encoded as a set of adjacency matrices with size $O[|V|^2]$. Memory requirements increase when storing parameters weights and threshold are included.

> **Theorem** *The size of weights needed for an threshold circuit to compute an arbitrary boolean function is upper bounded by* $o[\frac{(n+1)^{(n+1)/2}}{2^n}]$ *[9], [10]*

Theorem *The size of weights needed for a threshold circuit to compute an arbitrary boolean function is almost optimally lower bounded by* $O[\frac{(1/n)e^{-4nlog(3/2)}n^{n/2}}{2^n}]$ *[34].*

Both bounds lie between $O(2^n)$ and $O(n!)$, taking up $O[nlog(n)]$ bytes to encode in the worst case. Since the lower bound has been shown to be almost optimal, it is clear that strict bounds on the required weight size for computing arbitrary boolean functions cannot not offer more immediate insight to choosing weight sizes. A branch of related research focuses instead on finding weight bounds needed for close approximations instead. A general constructive result has been shown for smaller bounds of networks computing ε approximations of arbitrary boolean functions.

Theorem *For any arbitrary Boolean function* B*, there exists a threshold circuit which* ε *approximates* B *with maximal weight of at most* $\sqrt{n} * 2^{\tilde{O}(1/\varepsilon^2)}$ *[25].*

Although exponential in $\frac{1}{\varepsilon}$, this bound provides a tractable upper limit for 'reasonably close' approximations than the two theorems provided above. While it has not been shown to be optimal, a tight lower bound is still conjectured to be exponential with respect to $\frac{1}{\varepsilon}$. To see how well this bound translates to practical problems, we consider the most accurate networks trained on the CIFAR-10 and CIFAR-100 datasets, each having around 4% and 25% error respectively [61].

A boolean functional encoding of the classification of these two datasets by may be constructed by representing the input in 24-bit encoding (input size = $32 * 32 * 24 = 24576$) and having 7 different threshold networks with their outputs concatenated to denote the output class. Given this encoding, an approximation error of at most 25% on CIFAR-100 requires each of the four networks to achieve at least a 96% accuracy ($0.75^{1/7} \approx 0.96$). This gives $\varepsilon \approx 0.04$, which means the maximal weight required to compute each of the 7 networks would be $\sqrt{24576} * 2^{625}$ or around 625 bits to encode. Reaching an approximation error of at most 4% on CIFAR-10 requires 4 threshold circuits having at most 1% error computing their bits ($0.96^{1/4} \approx 0.99$), requiring each network to have a maximal weight of $\sqrt{24576} * 2^{10000}$ taking around 10000 bits each to encode.

Hardware limitations are not the only flaw of large weights, using unbounded weights causes problems such as overfitting, vanishing/exploding gradient, and catastrophic cancellation [22]. Although certain activations such as ReLU units help to remedy the vanishing/exploding gradient and catastrophic collision problems, they are still prone to overfitting with large weights. In practice, regularization and normalization are used to limit weight size so they are within the range of standard data types.

Size and depth are also heavily regulated feature of neural networks. For the majority of cases, a network will have a fixed architecture throughout its lifespan.

Theorem: *Any boolean function can be represented by a depth 2 threshold circuit.*

Theorem: *Any continuous function in any bounded domain has a depth two sigmoid network that can approximate it arbitrarily closely [11].*

These theorems do not provide tractable upper bounds on the size of the network and in particular, it has been shown that many functions with a high dimensional domain are not computable in polynomial size and depth 2 [27]. Increasing depth is an option for reducing the size requirement for approximation of various functions but by how much is still very much an open problem.

[53] How large do networks actually have to be? Iteratively pruned networks have similar accuracy to much larger networks however, training smaller networks from scratch does not yield the same result. This highlights a possible deficiency in the modern training algorithms deployed.

Empirical results and theory do not match up in many cases. The bounds that theory require for approximating arbitrary functions is much larger than the values seen in applications as was shown to be the case with weights, depth, size, and activations. [60] The space of functions computer by neural networks is special, they are similar to a class of functions studied in physics and it just so happens that the deep architectures of neural networks can compute them with small weight and depths.

In practice, the distinction between the two types of networks is quite different. It has been shown that for a large number of tasks, feedforward models actually perform just as well or better than their recurrent baselines [54]. There have been many attempts at explaining why this could be the case and all of them have to do with the speculation that tasks which utilized cyclic models may not really need the long dependencies that cyclic models offer. However, the sampled tasks included language modeling, speech synthesis, and machine translation which all seem like they should rely heavily on long dependencies. These results are intriguing as theory tells us that recurrent models should be much stronger than acyclic feedforward ones. A possible explanation for this phenomenon is that there are flaws with the method of training RNNs (namely backpropagation through time) that are not utilizing the cyclic structure of these networks to their full potential. This has been corroborated by a study showcasing that an n-gram model of length 13 performs as well as an unbounded context LSTM [55].

Sequential cyclic networks introduce a form of persistent memory over the length of inputs it receives. This memory is represented in the form of persistent neuron states and less commonly, variable delays in synapse and neuron computations. As it turns out, this memory

allows RNNs to model more conventional methods of computation such as Turing machines enabling an insertion of various network architectures into the computational complexity hierarchy forming a very loose ordering.

> **Theorem** *Integral weighted threshold circuits can simulate real weighted threshold circuits without an increase in size or depth or the number of bits needed to represent the weights [1].*

> **Theorem** *Recurrent integral weighted threshold circuits are equivalent in computational power to finite state machines [29], [32].*

> As opposed to networks utilizing continuous or smooth sigmoid activation functions:

> **Theorem** *An arbitrary Turing machine can be reduced in linear time to a recurrent network utilizing linear sigmoid activation functions and rational weights [17].*

> **Theorem** *There exists a super-Turing function which may be computed by a polysized real weighted recurrent network with sigmoid activations [21], [38].*

These four theorems showcase the varying degrees of sensitivity networks have to the domain of their weights with respect Networks with discrete activation functions are not as sensitive to the domain of their weights as shown by the four theorems below:

Representations of Power:

Analogously to hierarchies on the circuit classes AC^0 and NC^0 in complexity theory, network hierarchies are a means of separating networks based on order of increasing complexity. McCulloch and Pitts proposed ordering MP networks according to the longest cycle existing in the net is one such example. The goal of these orderings is to distinguish between the 'powerful' models from the 'weak' ones. In recent years an emerging branch of study known as Vapnik-Chervonenkis (VC) theory has shown some promising results in the construction of an ordering on specific types of networks.

VC theory introduces the concept of the VC dimension which describes a hierarchy on neural nets based on their structures and trainable parameters. The VC dimension of a neural net N is the largest dimension of some domain D' for which every mapping of $h : D' \rightarrow \{0, 1\}$ can be represented by N through some fixed setting of its parameters [18].

The VC dimension of a neural net gives a probabilistic upper bound on the expected test error of the network. Specifically:

$$Pr(\textit{test error} \leq \textit{training error} + \sqrt{\tfrac{1}{N}[D(log(\tfrac{2n}{D}) + 1) - log(\tfrac{\eta}{4})]}) = 1 - \eta$$

Where D is the VC dimension of the network, $0 \leq \eta \leq 1$, and N is the size of the training set, with the restriction that $D \ll N$.

This theorem is reinforced intuitively by Occam's Razor which states that when two models describe the same phenomenon equally well, the simpler of the two is usually better. In particular, we see from the theorem that a neural net with a higher VC dimension (more complex) will have a higher test error upper bound indicating a tendency to overfit. Whereas a simpler network with lower VC dimension which performs just as well classifying the training set will have a lower test error upper bound. An analogy would be preferring a lower degree polynomial regression model over a higher degree one in the case of when the error of the low degree polynomial is acceptably small.

It has been shown that many cases whose solutions were well approximated by larger and deeper models required them only because shallower models failed to closely approximate on the training data while maintaining a low VC dimension. Since this theorem only generates an upper bound, this is not always the case and while generally a good guideline, there are examples in which it is not the optimal strategy [19]. Choosing the right model for a problem is a balance between power versus the ability to well approximate the training set.

The VC dimension of a neural net also gives an upper bound on the number of functions it is able to approximate in a domain of higher dimension.

Sauer's Lemma *If d is the VC dimension of some family of functions F, then for any finite subset $X \subset D$, at most $\sum_{i=0}^{d} C(|X|, i) \leq |X|^d + 1$ functions from $X \rightarrow \{0, 1\}$ can be computed by functions in F*

Sauer's lemma by itself does not provide much information when the dimension of the desired target domain is close in size to the VC dimension of the network due to its nature as an upper bound, but for $|X| \gg d$, it provides a good initial indication of whether the network is likely to be capable of learning an approximation function.

The VC dimension is not a foolproof way to determine the computational power of a network however. For one, it is not known to be computable for a large portion of network architectures. Secondly, the known sample complexity derived from the VC dimension is sometimes exponential with respect to network size or depth [49] making the result trivial when either property becomes sufficiently large.

Theorem *All boolean functions computable in* $O(n)$ *can be computed by a threshold circuit of* $O(n)$ *depth and* $O(n^2)$ *size [48].*

The computational power of a network given by the above theorem will have extremely high depth and size; when training is considered on such an architecture, the space of functions it is capable of representing is also very high. To put things into perspective, a 24-bit encoding of the CIFAR-10 dataset would have $n = 32 * 32 * 24 = 24576$ bits required to encode each image. The required threshold function to classify this dataset would have depth $d = 24576$ and size $s = 603979776$ which almost seem like trivially large bounds.

A depth 13 sigmoid convolutional network with 3x3 kernels can achieve $\geq 95\%$ accuracy on the CIFAR-10 dataset [58]. Assuming the same 24-bit encoding of the image, each convolutional layer can be substituted with a fully connected layer of size $s = 24576 * max(32/3) = 2973696$. Using the previous theorems, we see that it is possible to construct a computationally equivalent threshold circuit of depth 13 and size $13 * 2973696 \approx 38658048$. These parameters (especially depth) are much smaller than the upper bound optimal circuit for this problem. This discrepancy between the size required for a reasonable approximation versus an optimal architecture highlights a key problem between theory and application. In terms of networks, it seems that speaking of strict upper and bounds for optimal networks is almost always impractical. Additionally, large networks will require increasingly large data set during training to ensure that overfitting does not occur. Many new problems are introduced and existing problems exacerbated when using conventional methods of training for such large and deep networks such as vanishing/exploding gradient, irregular loss surface, getting stuck in local minima, overfitting, etc.

Other ways of measure the complexity of networks is by analyzing the range of transformations they can perform on their inputs. There have been many other directions in this field of research including topological studies [56], trajectory length [57], and statistical learning theory. Throughout this paper, we focused on just one of the tenants of statistical lear

- VC dimension bounded by number of edges
- Reductions to TC^0
- From point of view of statistical learning theory

Conclusion:

Computational complexity is a complicated field, with many barriers which need to be overcome before breakthrough results are available. Applied results complicate the field further with large and deep systems somehow giving reasonable solutions to their proposed problems with much less resources than conjectured lower bounds might indicate. As a response to this phenomenon, recent trends in the field has split off from complexity theory's paradigm of

computing absolute or even approximate bounds on arbitrary functions and instead focusing on isolating and revealing properties of functions that compose the landscape of AI and NLP problems as well as finding novel ways to express the computational complexity of various network architectures.

Open problems:

Does there exist a general method to determine the depth required to compute or approximate an arbitrary boolean function with a threshold circuit of polynomial size

Does there exist a method to determine the size of the parameters required to compute or approximate an arbitrary boolean function

Is there any way to generalize results about fixed size circuits to convolutional ones?

Is there a difference in power between neural networks using different activation functions? (e.g. threshold versus sigmoid, ReLU vs sigmoid, etc)

Does a complete hierarchy on TC^0 exist?

How much does increasing the depth of a given neural network reduce the maximal weight or size required for some function requiring large weights and size with a lower depth.

Citations:

1. I Parberry. (1992) “On the complexity of learning with a small number of nodes”. In *Proc. 1992 International Joint Conference on Neural Networks*, volume 3, pages 893-898.

2. W McCulloch and W Pitts. (1943) "A Logical Calculus of Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics.*

3. F Rosenblatt. (1958) “The perceptron: A probabilistic model for information storage and organization in the brain”. *Psychological Review,* volume 65, pages 386-408.

4. M Minsky, S Papert. (1969) “Perceptrons: An Introduction to Computational Geometry”. *MIT Press, Cambridge, MA, USA*

5. A Beam. (2017) “Deep learning 101”. *Machine Learning and Medicine,* beamandrew.github.io/

6. G.E. Hinton, S. Osindero, and Y.-W. Teh. (2006) “A fast learning algorithm for deep belief nets”. *Neural Comp*. 18, 1527–1554

7. [http://image-net.org/challenges/LSVRC/2012/results.html]

8. J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. (2012) “Large scale distributed deep networks”. *In NIPS*, pp. 1232–1240

9. I Parberry. (1994) “Circuit Complexity and Neural Networks”, *MIT press, Cambridge, MA USA*

10. S. Muroga, I. Toda, and S. Takasu. (1961) “Theory of majority decision elements”. *Journal of the Franklin Institute* 271, 376–418.

11. G. Cybenko. (1989) "Approximations by superpositions of sigmoidal functions", *Mathematics of Control, Signals, and Systems*, 2(4), 303-314.

12. H. Mhaskar, Q. Liao, and T. Poggio (2016) “Learning Real and Boolean Functions: When Is Deep Better Than Shallow”. *Massachusetts Institute of Technology CBMM Memo No. 45.*

13. S. Edwards. (2003). “Making cyclic circuits acyclic”. In *Design Automation Conference (DAC),* ACM.

14. J. Ba and R. Caruana. (2014) “Do deep nets really need to be deep?”. In: *Advances in neural information processing systems*. 2654–2662

15. V.r Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. (2016) “Asynchronous methods for deep reinforcement learning”. *In Proceedings of the 33nd International Conference on Machine Learning,* ICML 2016, New York City, NY, USA, June 19-24, 2016, pages 1928–1937

16. J. Bruck. (1990) “On the convergence properties of the Hopfield model”. *Proc IEEE* 78(10):1579–1585

17. H Siegelmann and E Sontag. (1992) “On the computational power of neural nets”. *Journal of Computer and System Sciences, 50, 132-150.*

18. P.L. Bartlett and W. Maass, “Vapnik-Chervonenkis dimension of neural nets”, in *Handbook Brain Theory Neural Networks*, M.A. Arbib Ed. MIT Press, second edition. (2003) 1188–1192.

19. E. Baum and D. Haussler. “What size net gives valid generalization?” *Neural Computation,* 1(1):151-160, 1989.

20. Y. Bengio. “Practical recommendations for gradient-based training of deep architectures”. In K.-R. Muller, G. Montavon, and G.B. Orr, editors, *Neural Networks: Tricks of the Trade.* Springer.

21. H.T. Siegelmann and E.D. Sontag. (1994). “Analog computation via neural networks”, *Theoretical Computer Science,* 131:331-60.

22. http://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

23. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. (1986). “Learning internal representations by backpropagating errors”. *Nature*, 323:533-536.

24. W. Maass, G. Schnitger, and E.D. Sontag, “On the computational power of sigmoid versus Boolean threshold circuits”, in *Proc. 32nd IEEE Symp. Foundations of Computer Science.* (1991) 767-776

25. R. Servedio. (2006). “Every linear threshold function has a low weight approximator”. In *Proc. 21st Ann. IEEE Conference on Computational Complexity.*

26. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Sciences*, 792554-2558, April 1982.

27. A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. In *28th Annual Symposium on Foundations of Computer Science*, pages 99-110. IEEE Computer Society Press, October 1987.

28. A.M. Turing (1936). “On computable numbers, with an application to the Entscheidungsproblem”, *Proc. London Math. Soc*. (2), 42, pp. 230–265. *Corrections in Proc. London Math. Soc*. (2), 43 (1937), pp. 544–546.

29. S.C. Kleene. (1956). "Representation of events in nerve nets and finite automata", *Automata Studies, Princeton*, pp. 3-41

30. D.O. Hebb. (1949). “The organization of behavior”. *New York: Wiley & Sons.*

31. H.D. Block. (1970). "A review of 'Perceptrons an introduction to computational geometry' by Marvin Minsky and Seymour Papert". *Information and Control*, 501-522.

32. G. Piccinini. (2004). “The first computational theory of mind and brain: a close look at McCulloch and Pitts’s ‘Logical calculus of ideas immanent in nervous activity’”. *Synthese*, Volume 141, Issue 2, pp. 175-215

33. H.A. Simon. (1995). “Artificial intelligence: An empirical science”. *Artificial Intelligence*, 77.95-1 27.

34. J. Håstad. (1994). “On the size of weights for threshold gates”. *SIAM Journal on Discrete Mathematics* 7(3), 484–492

35. M. Goldmann, J. Håstad, and A. Razborov. (1992). “Majority gates vs. general weighted threshold gates”. *Computation Complexity 2*

36. P. Orponen. (1994). “Computational complexity of neural networks: A survey”. *Nordic Journal of Computing, 1*, 94-100.

37. M.D. Riedel and J. Bruck. (2012). “Cyclic boolean circuits”. *Discrete Applied Math*. 160(13-14).

38. H.T. Siegelmann. (1993). “Foundations of recurrent neural networks”. *Unpublished doctoral dissertation, Rutgers University*

39. J. Li, Y.F. Diao, M.D. Li, and X. Yin. (2009). “Stability analysis of discrete Hopfield neural networks with the nonnegative definite monotone increasing weight function matrix”. *Discrete Dynamics in Nature and Society*

40. S. Dieleman, J.D. Fauw, and K. Kavukcuoglu. (2016). “Exploiting cyclic symmetry in convolutional neural networks”. *arXiv preprint* arXiv:1602.02660.

41. T. Ben-Nun and T. Hoeer. (2018). “Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis”. *arXiv preprint* arXiv:1802.09941

42. V. Beiu, J. Peperstraete, J. Vandewalle, and R. Lauwereins. (1993). “Close approximations of sigmoid functions by sum of steps for VLSI implementation of neural networks”. *Proc. Romanian Symp. on Computer Science (Jassy, Romania 1993)* pp 31-50

43. X. Pan and V. Srikumar. (2016). “Expressiveness of rectifier networks”. In *ICML*, pages 2427–2435.

44. G.-B. Huang, Q.-Y. Zhu, K.Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan. (2006). “Can threshold networks be trained directly?” IEEE Trans. Circuits Syst. II, vol. 53, no. 3, pp. 187–191, Mar.

45. V. Beiu and J. G. Taylor. (1996). “On the circuit complexity of sigmoid feedforward neural networks”. *Neural Networks*, vol. 9, pp. 1155–1171, Oct.

46. S. Malik. (1994). “Analysis of cyclic combinational circuits”. *IEEE Trans. Computer-Aided Design,* vol. 13, pp. 950–956, July

47. Ronen Eldan and Ohad Shamir. (2015). “The power of depth for feedforward neural networks”. *arXiv preprint* arXiv:1512.03965

48. R. Livni, S. Shalev-Shwartz, and O. Shamir. (2014). “On the computational efficiency of training neural networks”. *In Advances in Neural Information Processing Systems*, pages 855–863.

49. Noah Golowich, Alexander Rakhlin, and Ohad Shamir. (2017). “Size-independent sample complexity of neural networks”. *arXiv preprint* arXiv:1712.06541.

50. M. Riedel. (2003). “Cyclic Combinational Circuits”. *PhD thesis, California Institute of Technology.*

51. M. Goldmann and M. Karpinski. (1993). “Simulating threshold circuits by majority circuits”. Proc. *ACM Symp. Theory of Computing* STOC 93, San Diego, CA, USA, May 16-18, 1993, pp. 551–560.

52. H.T. Siegelmann. (1996). “Recurrent neural networks and finite automata”. *Journal of Computational Intelligence*, 12(4), 567–574.

53. G. Castellano, A. Fanelli, and M. Pelillo. (1997). An iterative pruning algorithm for feedforward neural networks. IEEE Transactions on Neural Networks 8: 519–531.

54. http://www.offconvex.org/2018/07/27/approximating-recurrent/

55. C. Chelba, M. Norouzi, and Samy Bengio. (2017). "N-gram language modeling using recurrent neural network estimation". *arXiv preprint* arXiv:1703.10724.

56. B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. (2016). "Exponential expressivity in deep neural networks through transient chaos". *In Advances In Neural Information Processing Systems (NIPS)*, pages 3360–3368.

57. M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. (2017). "On the Expressive Power of Deep Neural Networks". *arXiv preprint* arXiv:1606.05336.

58. S. Hossein, H. Pour, M. Rouhani, M. Fayyaz, and M. Sabokrou. (2018) Let's keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint* arXiv:1608.06037

59. R. Basri, D. Jacobs. (2016). "Efficient Representation of Low-Dimensional Manifolds using Deep Networks". *arXiv preprint* arXiv:1602.04723

60. H.W. Lin, M. Tegmark, and D. Rolnick. (2017). "Why does deep and cheap learning work so well?" *Journal of Statistical Physics 168,* 1223–1247.

61. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

62.

Appendix

- **Backpropagation network:** Any network where backpropagation is a viable method for training. Minimum requirements being all activations have non-zero continuous first order derivatives (usually lipschitz continuous).

- **Cyclic Network:** A network is said to be acyclic if no edges exist such that a cycle forms between any two neurons. Or more intuitively, no neuron of the network has an output which displays recursion.

- **Combinational Input:** Also known as Time Independent Input, defines a type of computation a neural network can perform. Networks with this type of computation produce outputs only dependent on the current input. An example would be a fully connected feedforward neural network where each input is mapped to a unique output.

- **Hopfield Network:** A subtype of RNNs, fully connected, with either symmetric or antisymmetric weights, and boolean threshold activations.

- **Linear Function:** Any linear function may be expressed as $f : R^n \rightarrow R$ where $f(x_1, ..., x_n) = \sum_{i=1}^{n} w_i x_i$ for $w_i \in R$

- **Linear Threshold Function:** A linear threshold function is any function which can be represented as the truth value of the statement that a linear function $f : R^n \rightarrow B$, $f(x_1, ..., x_n) \geq h$ for some $h \in R$

- **Recurrent Neural Network:** Symmetric, ReLU or tanh activation, sequential input, updates in layers temporally.

- **Separability:** A linear function $f : R^n \rightarrow B$ with threshold h is called δ separable if $\forall (x_1, ..., x_n)$ such that $f(x_1, ..., x_n) < h$, $f(x_1, ..., x_n) \leq h - \delta$, $\delta \in R$.

- **Sequential Input:**

- **Synchronicity:** Refers to the order and number of neurons or weights updated at the same time in a network. Asynchronous updates are strictly one update per time step while synchronous updates may refer to any number of updates per time step.

- **Synchronous layer update:** The synchronous update process in which the set of neurons updated at time t are the neurons with afferent neurons which were active at time $t - 1$. The set of neurons active at time 0 are exactly the input neurons.

- **Threshold Circuits:** A threshold circuit C is defined to be the 5-tuple (V, X, Y, E, l) where
    - V is a finite ordered set
    - $X, Y \subseteq V$
    - (V, E) is a directed acyclic graph
    - $l : V \rightarrow \Theta$ where Θ is the set of all linear threshold functions
- **Uniform:** A family of circuits is said to be uniform if there exists a Turing Machine T such that T outputs the n^{th} circuit in the family when given the unary string 1^n as input.
- **Vapnik Chervonenkis Dimension:** The VC dimension for some parameterized class of functions F where $f(\theta, x) \in F$, $f(\theta, x) : x \rightarrow \{0, 1\}$, $x \in D$ is defined to be the size of the largest subset $D' \in D$ such that
    - $\forall h : D' \rightarrow \{0, 1\} \ \exists \theta \ \forall x \in D' (f(\theta, x) = h(x))$