CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 114 **July 24, 2022**

From Associative Memories to Deep Networks

Tomaso Poggio

Abstract

Associative memories were implemented as simple networks of threshold neurons by Willshaw and Longuet-Higgins in the '60s. Today's deep networks are quite similar: they can be regarded as approximating look-up tables, similar to Gaussian RBF networks. Thinking about deep networks as large associative memories provides a more realistic and sober perspective on the promises of deep learning.

Such associative networks are not powerful enough to account for intelligent abilities such as language or logic. Could evolution have discovered how to go beyond simple reflexes and associative memories? I will discuss how inventions such as recurrence and hidden states can transform look-up tables in powerful computing machines.

In a more recent update I will outline a framework describing how deep networks may work, including transformers. The framework is based on proven results plus a couple of conjectures – still open.

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

From Associative Memories to Deep Networks

Tomaso Poggio

Abstract

Associative memories were implemented as simple networks of threshold neurons by Willshaw and Longuet-Higgins in the '60s. Today's deep networks are quite similar: they can be regarded as approximating look-up tables, similar to Gaussian RBF networks. Thinking about deep networks as large associative memories provides a more realistic and sober perspective on the promises of deep learning.

Such associative networks are not powerful enough to account for intelligent abilities such as language or logic. Could evolution have discovered how to go beyond simple reflexes and associative memories? I will discuss how inventions such as recurrence and hidden states can transform look-up tables in powerful computing machines.

1 Introduction

The starting observation is that associative memories such as Willshaw nets and most deep networks are essentially look-up tables that, depending on hyperparamers, may be able not only to retrieve training examples but also to perform some simple interpolation among similar training examples – called generalization in machine learning. I will argue that associative memories of this kind are not powerful enough to underlie the intelligence needed in language and logic and ask how evolution may have discovered the step beyond memory towards intelligence. In particular, I will describe a way, among many other possibilities, to transform a look-up table into a powerful computing machine..

2 Willshaw Nets

Holograms store information in the form of an optical interference pattern recorded in a photosensitive optical material. Light from a single laser beam illuminates a noise-like reference image (originally generated from ground glass) as well as the pattern to be stored, yielding an interference pattern stored in the hologram. Many thousand such pairs of associations can be recorded on a single hologram. Each stored data can then be read-out from the hologram by using as input its associated reference pattern.

An associative memory $A_{X,Y}$ can be modeled as a one layer "shallow" network [1] storing the correlation matrix between input and output. Figure 1 shows the training phase of the network. In the read-out phase, the output y can be retrieved by inputting the associated x into the

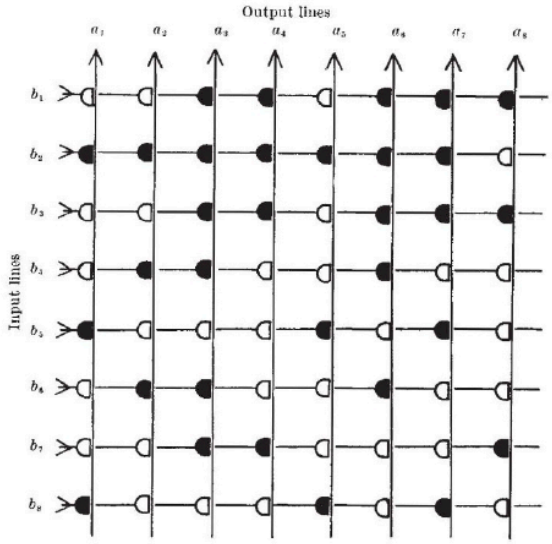


Fig. 4. An associative net.

Figure 1: *A original figure from Willshaw et al. [1] showing an associative memory network. The matrix of connections correspond to the matrix W of weights in a shallow network, that is $a = Wb$ where a and b are the output and the input vectors respectively and W is the matrix of weights. In the text we use y instead of a and x instead of b.*

network, that is by computing $A_{X,Y} \circ x$ (Willshaw computed $R \circ A_{X,Y} \circ x$ where R represents a set of thresholds on the outputs to improve accuracy retrieval in a otherwise linear network).

The basic intuition is explained by a simple linear model. Suppose that we want to associate each pattern $y_n, \quad n = 1, \cdots, N$ to a noise-like key vector x_n, where $x, y \in \mathbf{R}^D$ and in this simple example $N = D$. The noise-like assumption on the x_n is equivalent to assuming that $X^T X \approx I$, where X is the matrix of all the inputs (x_n are the columns of X). The optimal least-square solution of the equation $AX = Y$ is $A = YX^T$ if the columns or the rows of X are orthonormal. Thus, to retrieve y_i, it is sufficient to input the key x_i to the network and get $Ax_i \approx y_{i,j}\delta_{i,j} = y_i$. In dealing with binary vectors, this linear associative network can be improved by using thresholds to clean up the output as Willshaw did.

2.1 Networks: shallow, deep and recurrent

Willshaw experimented "de facto" with multilayer networks when he found that a recurrent version of his one layer network performed quite well. In any case, this is still a one layer network, quite different from modern multilayer networks. It turns out that Willshaw experimented "de facto" with multilayer networks when he found that a recurrent version of his one layer network performed quite well. As he reported *"...it was found by computer simulation...that the initial response to a given cue could be improved by feeding the output back into the associative net and continuing until the sequence of outputs so generated converged onto a single pattern..."*. Furthermore, *"The same "cleaning-up behavior" was seen when patterns were stored in sequence. Pattern A was associated with B. B with C. C with D. and so on, the last pattern being stored with A. When a fragment of A was used as a cue and then the output used as the next input, after a few passes the sequence of retrieved patterns converged onto the stored sequence, even when the initial cue was a very poor representation of one of the stored patterns. Simulation experiments were performed to see what cycle of outputs would result from any arbitrarily selected cue. (Because each input determines the next output and there is only a finite number of possible outputs, the sequence of outputs must eventually lead into a cycle.).."* Of course a recurrent network is just a multilayer network with shared weights across different layers[2]. In any case, the ideas of single layer as well as recurrent associative networks – as well as their implementations – were alive and well fifty years ago!

Let us now consider a deep network written as

$$f(x) = (V_L \sigma(V_{L-1} \cdots \sigma(V_1 x))) \tag{1}$$

where *sigma* is the RELU nonlinearity. The equation can also be rewritten as

$$f(x_j) = V_L D_{L-1}(x_j) V_{L-1} \cdots\cdots V_{k+1} D_k(x_j) V_k \cdots D_1(x_j) V_1 x_j \tag{2}$$

where $D_k(x_j)$ is a diagonal matrix with 0 and 1 entries depending on whether the corresponding RELU is active or not for the specific input x_j, that is $D_{k-1}(x_j) = diag[\sigma'(N_k(x_j)]$ with $N_k(x_j)$ the input to layer k.

The claim is that deep and recurrent networks can be regarded as stacked associative one-layer networks of the Willshaw type and they perform the same basic computation, just a bit more than a look-up table. In this view depth and recurrence increase retrieval performance but do not change the computational power.

This conclusion is reinforced by the following argument. An example of an "interpolating" look-up table is a RBF network with Gaussian units (see Figure 2.1). A Gaussian unit computes $e^{-\frac{(x-x_i)^2}{\sigma^2}}$ where x_i is the "center" of unit i. Increasing σ changes the network from a look-up table kind of memory, that recognizes only the training data, to a "learning" system that combines a few examples similar to each other and thus "generalizes". It turns out that under certain training conditions (e.g. starting with "largish" norms for the matrices weights) a deep network converges to a set of weight matrices that corresponds to a standard kernel machine with the so called NTK kernel[3], which is quite similar[4] to a radial kernel of the Laplacian type. A similar message – many neural networks are similar to look-up tables – is conveyed by the tale of NetTalk, one of the very first success stories of neural networks in the 80s. NETtalk learned to pronounce written English text by being shown text as input and matching phonetic transcriptions for comparison. After the initial claims it turned out that a nearest-neighbor classifier could do as well. The boundary between associative networks – shallow or deep — and learning networks is very thin, since the underlying machinery is very much the same and the difference is just in parameter values.

3 Evolution: from associative memories to computing machines?

3.1 Look-up tables cannot support the kind of intelligence required for language and logic

Clearly human intelligence is not just one associative memory. Thus intelligence is not just one deep learning network trained end-to-end. Even if we accept that associative networks are an important part of how we think, from visual and speech recognition to Kahneman's System One, we still have to explain System Two which is more deliberative, and more logical.

3.2 From memory to powerful computing machines

The question then is: is it possible to build powerful computing machines from a set of look-up tables? A positive answer would make easier to understand how intelligence may have evolved from simple associative reflexes.

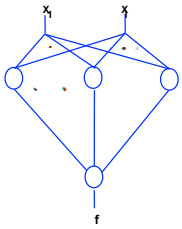
It turns out that a finite state machine (think of it as a Turing machine that can run only T steps) can be synthesized from associative modules containing the instruction set of the machines, that is a memory mapping states and inputs into states and outputs. From this point of view a *recurrent* network *with state variables* is a finite state machines with the weights representing the look-up table with the program. Thus it may have been quite easy for evolution to go from associative modules to computing machines. Another direction, among the many schemes equivalent to the finite state machine, is to think of recurrent associative networks as the basis

Radial Basis Function networks

$$\min_{f \in H} \left[\frac{1}{n} \sum_{i=1}^{n} V(f(x_i) - y_i) + \lambda \|f\|_K^2 \right]$$

implies $f(\mathbf{x}) = \sum_i^n \alpha_i K(\mathbf{x}, \mathbf{x}_i)$

equivalent to



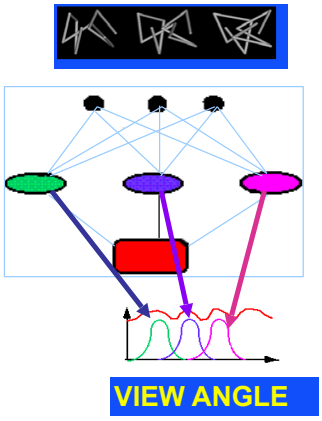


Figure 2: *A kernel machine with Gaussian radial basis functions is a look-up table for very small standard deviation of the Gaussian kernel. It is an approximating look-up table otherwise.*

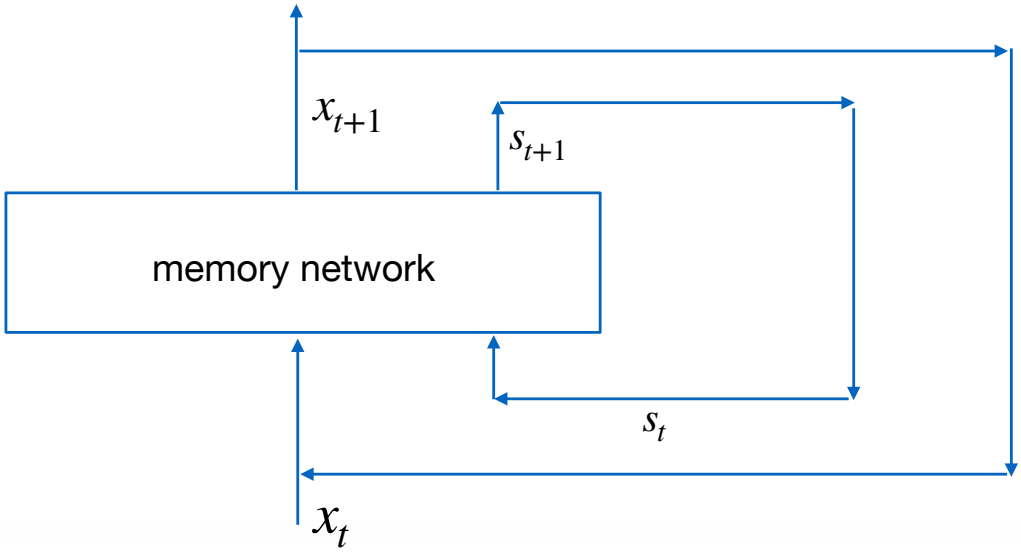


Figure 3: *A recurrent network consisting of one (or more) associative memory layers that map inputs and state $x(t), s(t)$ into outputs and new state $x(t+1), s(t+1)$ is the core of a computing machine roughly equivalent to a finite state machine.*

for computational capabilities similar to the Lambda calculus and LISP, along the lines described by Plate in his Holographic Reduced Representations[5].

At this point there are several questions we may be tempted to ask. Is the intelligence of a honeybee explainable in terms of a finite state machine? Did evolution discover the trick to a universal computing machine? Were a number of primitive routines implemented in fixed neural circuits? How did flexible calling of routines evolve? Did this step involve the ability to flexibly copy synaptic weights? There may be a more or less fixed set of routines and intelligence may have evolved the ability to use them in more and more complex program. Was language the key in this discovery or merely one of its main results? Are internal simulations a way to develop or learn new programs, for instance using virtual worlds the way Deep Mind has done with AlphaZero? And how did the ability to transmit information to future generations increase human intelligence[1]?

The challenge for neuroscience is to find out which circuits underly our intelligence and how are they different with respect to associative networks. I regard this as the core problem in our present quest to understand human intelligence and replicate it in machines.

[1] And the illusions we have about it.

4 Theoretical Framework: How Deep Nets May Work

I propose here in July 2022a theoretical framework that aims to explain how deep networks work as well as they do and what are the properties of different architectures.

The *key assumption* is about the world, that is about the tasks that networks can learn. The assumption -let me call it the *PL thesis* – is that all learnable functions must have a representation with the property of *compositional sparsity*, that is they can be represented as compositional functions with a function graph comprising constituent functions with a bounded – and "small" – dimensionality. The assumption is justified by a theorem stating that *all efficiently computable functions* are compositionally sparse.

- **Theorem 1** *For functions that are compositionally sparse, approximation is possible without incurring in the curse of dimensionality.*

  In the overparametrized square loss case, generalization depends on solving a sort of *regularized ERM*, that consists of finding minimizers of the empirical risk with zero loss, while selecting the one with lowest complexity. Recent work has provided theoretical and empirical evidence that this can be accomplished by SGD (with norm regularization under the square loss or without regularization under an exponential loss). The conjecture is then that this optimization problem can be solved if the graph of the underlying regression function

  1. is known and
  2. takes the form of a sparse graph, such as, for instance, a convolutional network.

  A further conjecture is that for dense networks the same problem cannot be solved using ℓ_2 minimization. Sparsity must be explicit in the architecture of the network for ℓ_2 minimization to work.

- The second part of our framework is about the case of unknown function graph and sparsity constraints in optimization. I propose the conjecture that when the sparse graph structure of the underlying regression function is not known, optimization with sparsity constraints such as ℓ_0 or ℓ_1 is needed. In particular, two situations should be considered. The main one is focused on transformers, the second on dense networks under sparsity constraints.

  For transformers the claim is that the self-attention layer finds a sparse representation of the input in a way similar to ℓ_0 minimization, assuming its existence. I will show that the stages of self-attention and MLP with normalization and residual connections can be seen as an iteration of IHT followed by an iteration of a one-layer MLP. The first set of iterations implements sparsity on $y = f(H_D(\bar{X}(\bar{X})^T)$ where $H_D(x) = xH(x)$ with H being a threshold on x. The second is a multilayer dense network – or a recurrent dense net – on a sparse input.

  For dense networks it is known that a CNN-like inductive bias can be learned from data and through training by using a modified ℓ_1 regularization. Consistent with this empirical finding, pruning of a dense network by using iterative magnitude pruning (IMP) also works.

In summary, the claim is that sparsity of the underlying regression function is the key assumption in machine learning. Sparsity then leads to sparsity-biased optimization techniques during optimization. This is the case of transformers. ℓ_2 techniques however can be used when the sparsity is known and implemented in the architecture of the network.

4.1 Selfattention selects a sparse support

What does self-attention do? We claim it finds a sparse representation of the input in a way similar to l_0 minimization. The transformers layers without MLP but with layer normalization and residuals are similar to iterations of IHT (Iterative Hard Thresholding), implementing sparsity by regressing on $y = f(H_D(\bar{X}(\bar{X})^T)$ where $H_D(x) = xH(x)$ where H is a (soft) threshold on x. A connection with associative memories may be found via the RIP property that is imposed on X via the matrices Q,K – that is $H_D(XW_QW_K^TX^T)$ should show RIP – and that corresponds to the "noiselike" signals of holographic memories.

In summary, we address the question of how do transformers deal with unknown function graphs. We propose that 1) they assume that the underlying function depends on an unknown subset of the input variables and is thus sparse (in fact compositionally sparse) and 2) they solve the constrained ℓ_0 problem

$$min||x||_0 \qquad s.t. \qquad b = Ax + \epsilon \tag{3}$$

by finding x and at the same time the matrices W_Q and W_K such that $A = QK^T$, with $Q = XW_Q$, $K = XW_K$, has the RIP property.

Furthermore our claim is that they do that by implementing a greedy sparsity algorithm of the Iterative Hard Thresholding type through several of the transformer layers (self-attention and MLPs) with residual connections.

4.1.1 Sparse solutions of $Ax = y$

In linear algebra, the restricted isometry property (RIP) characterizes matrices which are nearly orthonormal, at least when operating on sparse vectors. The concept, which is a formalization of the old concept of noiselike" signals used in the context of holographic memories, was introduced by Emmanuel Candès and Terence Tao. It has been shown that with exponentially high probability, random Gaussian, Bernoulli, and partial Fourier matrices satisfy the RIP property with a number of measurements nearly linear in the sparsity level.

Let A be an $m \times p$ matrix and let $1 \leq s \leq p$ be an integer. Suppose that there exists a constant δ_s such that, for every $m \times s$ submatrix A_s of A and for every s-dimensional vector y,

$$(1 - \delta_s)||y||_2^2 \leq ||A_s y||_2^2 \leq (1 + \delta_s)||y||_2^2. \tag{4}$$

Then, the matrix A is said to satisfy the s-restricted isometry property with restricted isometry constant δ_s. This condition is equivalent to the statement that for every $m \times s$ submatrix A_s of A we have

$$||(A_s)^T A_s - I_{s\times s}||_2 \leq \delta_s. \tag{5}$$

where $I_{s\times s}$ is the identity matrix and $||\cdot||_2$ is the operator norm. Finally this is equivalent to stating that all eigenvalues of are in the interval $[1-\delta_s, 1+\delta_s]$.

4.1.2 Iterative hard thresholding

The iterative hard thresholding algorithm is an iterative algorithm to solve the rectangular system $Az = y$, knowing that the solution is s-sparse. We shall solve the square system $A^T Az = A^T y$ instead, which can be interpreted as the fixed-point equation

$$z = (\mathbf{I}_d - A^T)z + A^T y. \tag{6}$$

Classical iterative methods suggest the fixed-point iteration $x_{n+1} = (I_d - A^T A)x_n + A^T y$. Since we target s-sparse vectors, we only keep the $(I_d - A^T A)x_n + A^T y = x_n + A^T(y - Ax_n)$ at each iteration. The resulting algorithm reads as follows:

$$x^{n+1} = H_D(x^n + A^T(y - Ax^n)) \tag{7}$$

4.1.3 Transformers

4.2 K, Q, V

$X \in \mathcal{R}^{T,d_{in}}$; $Q = XW_Q$ with $W_Q \in \mathcal{R}^{d_{in},d_k}$; $K = XW_K$ with $W_K \in \mathcal{R}^{d_{in},d_k}$; $V = XW_V$ with $W_V \in \mathcal{R}^{d_{in},d_{out}}$

The matrix $XW_Q W_K^T X^T \in \mathcal{R}^{T,T}$ can be a RIP matrix with appropriate choices of W_K, W_Q.

Notice that the standard formulation of the transformer layers can be written as

$$y = x + MLP(LayerNorm(x + Attention(LayerNorm(x)))$$

.

This implies that the sparsity function and the nonlinear association are intertwined. There is however a parallel formulation with similar empirical performance (see PALM paper) which can be written as

$$y = x + MLP(LayerNorm(x)) + Attention(LayerNorm(x))$$

which is easier to analyze in our framework

4.2.1 Self-attention and IHT

For our goals consider the equation $Ax = y$. Computes as a first approximation $x^1 = H_D(A^T y)$, assuming A is a RIP matrix and thus $AA^T \approx I$. This gives $A^T Ax \approx A^T y$. Thus $x \approx H_D(A^T y)$. This is the first step in the iteration. $H_D(x)$ is the Donoho soft threshold $H_D(x) = xH(x)$, where H is the Heaviside step function.

Self attention in the first layer computes $H(XQK^TX^T)XW_V$ this is a slight extension of Donoho soft-thresholding (because of W_V).

The iterations in IHT correspond to successive layers in a transformer (it is important for this interpretation that there are residual connections).

4.2.2 Pruning

We remark that dense networks cannot learn convolution under L_2 minimization but can under L_1 minimization. In particular, the possibility of learning CNN-like inductive bias from data and through training was investigated in Neyshabur (2020). It was shown that training using a modified L_1 regularization is a way to induce local masks for visual tasks. Consistent with this finding, pruning of a dense network by using iterative magnitude pruning (IMP) on FCNs trained on a low resolution version of ImageNet uncovers [**?**] sub-networks characterized by local connectivity, especially in the first hidden layer, and masks leading to local features with patterns very reminiscent of the ones of trained CNNs. Notice that IMP is an iterative algorithm that converges under appropriate conditions to L_0 minimizers.

This is similar to our results on pruning: enforcing sparsity during training leads to structures characterized by locality. d'Ascoli et al. (2019) studies the role of CNN-like inductive biases by embedding convolutional architectures within the general FCN class. It shows that enforcing CNN- like features in an FCN can improve performance even beyond that of its CNN counterpart. Finally, Tolstikhin et al. (2021) shows that by considering a particular multilayer perceptron architecture, called MLP-mixer, some of the CNN features can be learned from scratch using a large training dataset.

Acknowledgments I am grateful to to Ron Rivest, Santosh Vempala and Owen Kunhardt for useful comments. This material is based upon work supported by the Center for Minds, Brains and Machines (CBMM), funded by NSF STC award CCF-1231216, and part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, 1969.

[2] Q. Liao and T. Poggio. Bridging the gap between residual learning, recurrent neural networks and visual cortex. *Center for Brains, Minds and Machines (CBMM) Memo No. 47, also in arXiv*, 2016.

[3] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On Exact Computation with an Infinitely Wide Neural Net. *arXiv e-prints*, page arXiv:1904.11955, April 2019.

[4] Ronen Basri, Meirav Galun, Amnon Geifman, David Jacobs, Yoni Kasten, and Shira Kritchman. Frequency Bias in Neural Networks for Input of Non-Uniform Density. *arXiv e-prints*, page arXiv:2003.04560, March 2020.

[5] T Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. *International Joint Conference on Artificial Intelligence*, pages 30–35, 1991.

[6] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *NIPS*, pages 1177–1184, 2007.

[7] T. Poggio and W. Reichardt. On the representation of multi-input systems: Computational properties of polynomial algorithms. *Biological Cybernetics, 37, 3, 167-186.*, 1980.

[8] Tim Salimans and Diederik P. Kingm. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems*, 2016.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[10] G Palm. On associative memory. *Biological Cybernetics*, 36:19–31, 1980.

[11] T. Poggio and Q. Liao. Generalization in deep network classifiers trained with the square loss. *CBMM Memo No. 112*, 2019.

[12] Paulo Jorge S. G. Ferreira. The existence and uniqueness of the minimum norm solution to certain linear and nonlinear problems. *Signal Processing*, 55:137–139, 1996.

[13] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Theory I: Why and when can deep - but not shallow - networks avoid the curse of dimensionality. Technical report, CBMM Memo No. 058, MIT Center for Brains, Minds and Machines, 2016.

[14] H.N. Mhaskar and T. Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, pages 829– 848, 2016.

[15] H. N. Mhaskar"and T. Poggio. Function approximation by deep networks, 2020.

[16] T. Poggio, F. Anselmi, and L. Rosasco. I-theory on depth vs width: hierarchical function composition. *CBMM memo 041*, 2015.

[17] Christos H. Papadimitriou, Santosh S. Vempala, Daniel Mitropolsky, Michael Collins, and Wolfgang Maass. Brain computation by assemblies of neurons. *Proceedings of the National Academy of Sciences*, 117(25):14464–14472, 2020.

[18] T Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19(4):201–209, 1975.

[19] Y. Han, G.Roig, G. Geiger, and T. Poggio. Scale and translation-invariance for novel objects in human vision. *Scientific Reports*, 10(1):1411, 2020.

[20] Adi Shamir, Odelia Melamed, and Oriel BenShmuel. The dimpled manifold model of adversarial examples in machine learning. *CoRR*, abs/2106.10151, 2021.

[21] Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian J. Goodfellow. Adversarial spheres. *CoRR*, abs/1801.02774, 2018.

[22] Partha Niyogi, Federico Girosi, and Tomaso Poggio. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE*, 86:2196 – 2209, 12 1998.

[23] Zhiyuan Li, Yi Zhang, and Sanjeev Arora. Why are convolutional nets more sample-efficient than fully-connected nets? *CoRR*, abs/2010.08515, 2020.

[24] Eran Malach and Shai Shalev-Shwartz. Computational separation between convolutional and fully-connected networks. *CoRR*, abs/2010.01369, 2020.

[25] W. Reichardt, T.Poggio, and K. Hausen. Figure-ground discrimination by relative movement in the visual system of the fly ii: towards the neural circuitry. biol. cybern. 46, 1-30. *Biological Cybernetics*, 46:1–30, 01 1983.

5 Appendix: old and new remarks, some related to the subject of the memo

5.1 Observations about depth

The solution of $AX = Y$ with $X \in \mathbf{R}^{D,N}$ is $A = YX^{\dagger} = Y(X^TX)^{-1}X^T$. When $D > N$ $X^{\dagger}X = I$ with the matrix $I \in \mathbf{R}^{N,N}$[2]. Thus $A = YX^{\dagger}$ is always the optimal solution in the least square sense of $AX = Y$. This also implies that minimization of square loss in a one layer network with weights W over a training set X, Y has a minimum for $W = YX^{\dagger}$. For $D \geq N$, the optimal solution for W_2 in a 3-layer network with $W_1 = X^T$ and $W_3 = Y$ is $W_2 = (X^TX)^{-1}$. Interestingly, the computation of W_2 could also be learned by a recurrent network that may be easier to train (see Appendix).

So far I have described linear networks. The RELU nonlinearity after unit summation can be added as follows. Let us assume a deep network written as

$$f(x) = (V_L\sigma(V_{L-1}\cdots\sigma(V_1x))) \tag{8}$$

where $\sigma(x) = \sigma'(x)x$, which captures the homogeneity property of the RELU activation. The equation can be rewritten for each training example as

$$f(x_j) = V_LD_{L-1}(x_j)V_{L-1}\cdots\cdots V_{k+1}D_k(x_j)V_k\cdots D_1(x_j)V_1x_j \tag{9}$$

where $D_k(x_j)$ is a diagonal matrix with 0 and 1 entries depending on whether the corresponding RELU is active or not for the specific input x_j, that is $D_{k-1}(x_j) = diag[\sigma'(N_k(x_j)]$ with $N_k(x_j)$ the input to layer k.

The presence of the $D(x)$ matrices makes the network more powerful in terms of approximating any continous functions instead of just linear functions. It also requires more than the simple linear analysis described above.

Remarks

- Consider instead of $W_{i,j} = (X^TX)_{i,j}$ the choice

$$W_{i,j} = K(x_i, x_j) = \sum_{\ell}^{\infty} \lambda_\ell\phi_\ell(x_i)\Phi_\ell(x_j) = \Phi^T(x_i)\Phi(x_j) \tag{10}$$

  where the (possibly infinite) column vector Φ is $\Phi(x) = \lambda_i^{\frac{1}{2}}\phi_\ell(x)$ and the λ_ℓ are the eigenvalues of the integral operator associated with K. A shift-invariant kernel such as the Gaussian kernel has $\phi_\ell(x)$ which are orthonormal Fourier eigenfunctions. It can be approximated by random Fourier features $e^{-i\omega x}$ with ω drawn from a Gaussian distribution [6].

[2] In the case of $D < N$ the solution $A = YX^{\dagger}$ with $X^{\dagger} = X^T(XX^T)^{-1}$ is still the best in the optimal square sense.

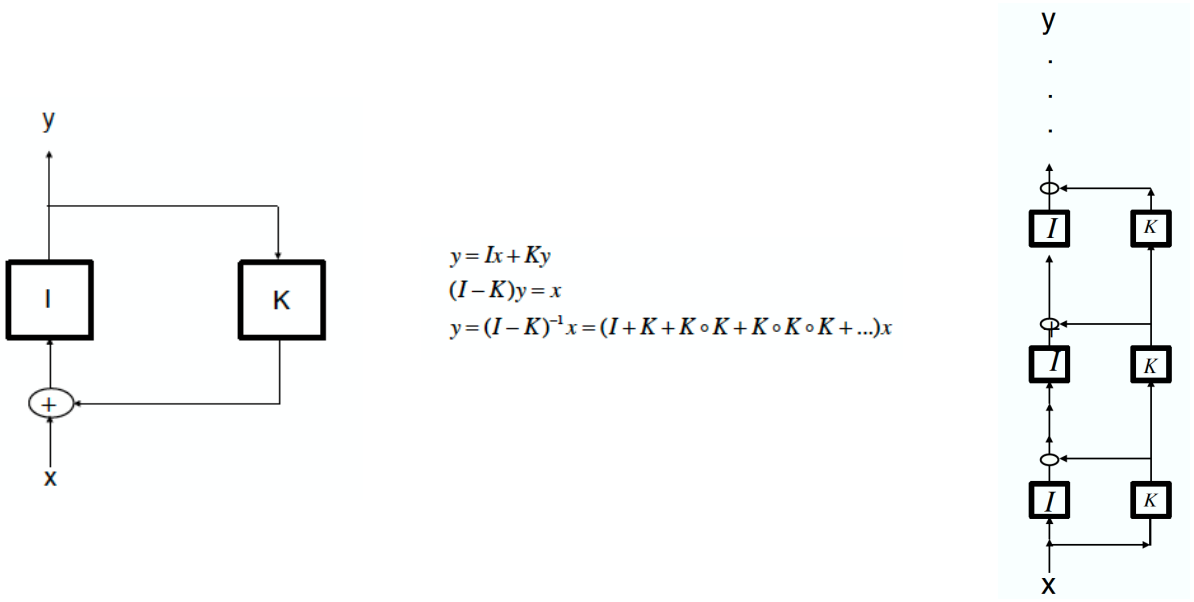


Figure 4: *Setting $K = I - X^T X$ allows the recurrent network as well as its unrolled deep network counterpart to compute $(X^t X)^{-1}$.*

- The "holographic" scheme of using a "noiselike" key vector associated with a signal is almost exactly the algorithm at the core of spread spectrum CDMA techniques used to encode and decode cell phones communication.

5.2 ResNets

Assume that the weight matrix of the recurrent network is learned to be

$$W_2 = (I - XX^T), \quad \forall i = 1, \cdots, L-1., \tag{11}$$

Since division of operators can be approximated by its power expansion, that is $\frac{I}{I-K} = (I + K + K^2 + \cdots)$, a recurrent network as shown in Figure 4 computes $(I-K)^{-1}$. If $K = I - XX^T$, the recurrent network computes $(XX^T)^{-1}$. Alternatively, a recurrent network can be replaced by a deep residual network (ResNet) of $L-1$ layers with the same K (see Figure and [7, 2]. Convergence requires the condition $||XX^T - I|| < 1$, which is usually satisfied if the weight matrices are normalized (for instance by weight normalization[8] or indirectly by batch normalization[9]). Estimates about retrieval errors in such associative memories and ways to reduce them by using thresholds are given in [1, 10].

Thus training a recurrent network under the square loss on a training set (X, Y) by unrolling it in L layers and imposing shared weights for the first $L-1$ layers should converge to the solution suggested by Equations.

Remarks

- The convergence of a recurrent network for $L \to \infty$ – where L is the number of iterations – is guaranteed by Brower's fixed point theorem if the operator $Tz = Wz$ is non-expansive, that is if $||Tx - Ty|| \leq ||x - y||$. The fact that the operator corresponding to the transformation of each layer of the network is non-expanding follows from the fact that $||Wz|| \leq ||W||||z||$, assuming that $||W|| = 1$ because of batch normalization(BN) (see [11] for the importance of BN). Notice that this holds for linear networks but also for networks with RELU nonlinearities. If the inputs x satisfy $||x|| \leq 1$ the set of fixed points of T contains a unique minimum norm element (see [12])

- Deep networks with $L - 1$ layers of identical input and output dimensionality and shared weights across layers are equivalent to a one-layer recurrent network run for $L - 1$ iterations. Empirically it seems[2] that non-shared weights give a small advantage despite the much larger number of parameters with respect to equivalent shared-weights networks. It is unclear why it is so. From this perspective, multiple layers may be required only to exploit the blessing of compositionality[13, 14]. In other words, depth main purpose may be to implement graphs where the nodes compose different functions[3].

5.3 Deep learning and signal processing

- The old associative networks assumed noise-like inputs that are approximately orthogonal (like in the original concept of holography implementing an associative memory), that is $x_i^T x_j = \delta_{i,j}$. A recent analysis [11] of deep network trained under the square loss identifies a bias towards orthogonality induced by normalization techniques such as batch normalization. Quasi-orthogonality makes it easy to invert a deep network as it is required in an autoencoder. Notions related to random projections and the Johnson-Lindestrauss lemma may also be relevant.

- The architecture of conv-nets, that is deep convolutional networks reflects a specific type of Directed Acyclic Graph (DAG). It turns out that all functions of several variables can be decomposed according to one or more DAGs as compositional functions, that is functions of functions[14]. Often such decompositions satisfy a hierarchical locality condition: even if the dimension of the overall function is arbitrarily high, the constituent functions are of small, bounded dimensionality. For these functions and these decompositions, approximation theory proves[14] that deep networks reflecting the underlying compositional DAG can avoid the curse of dimensionality, whereas shallow networks cannot. Convolutional networks are an example of this (locality of the kernel rather than weight sharing is the key property in avoiding exponential complexity, see [15]). Not accidentally, convolutional networks represent one of the main success stories of deep learning. Thus the main reason for deep networks as opposed to shallow, recurrent networks may in fact be to escape the curse of

[3] Often this implies some kind of "pooling" (even just by subsampling)

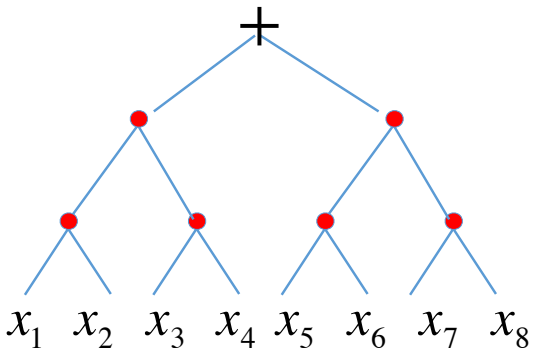


Figure 5: *The figure shows the graph of a function of eight variables ($f : \mathbf{R}^8 \to \mathbf{R}$)) with constituent functions of dimension two.*

dimensionality by exploiting compositionality, which can use (to reduce number of units) subsampling or "pooling", that is stages at which the outputs of constituents functions undergoes aggregation , as in Figure 5.

- Compositional architectures can be regarded as reflecting iterated functional relations of the kind "compose parts" as in $f(x_1, x_2, x_3) = f_1(f_2(x_2, x_3), f_3(x_3))$, where f_1 reflects the composition of f_2 and f_3 and f_2 composes x_1 and x_2. A deep associative network of this type is then closely related to what is called "hierarchical vector quantization (VQ)"[16]. The similarity is especially strong if we assume weight matrices that are derived from RBF kernels. This corresponds to memorizing, at the lowest level, the association of basic features and then the association of their associations (think of hierarchical JPEG encoding)[4].

- The claim that deep networks are quite similar to "linear" RBF networks is supported by recent results[3] on the Neural Tangent Kernel (NTK). It turns out that under certain training conditions (e.g. starting with "largish" norms for the matrices weights) a deep network converges to a set of weight matrices that corresponds to a standard kernel machine with the NTK kernel. Furthermore, classification performance is quite good – though not the best possible – and the NTK itself is equivalent[4] to a classical RBF kernel, the Laplacian.

- An alternative to deep networks as models of the brain are neural assemblies. The idea received new life from some recent very interesting work [17]. The obvious question is about

[4]Starting from a small number of primitive features, there is a hierarchy of more complex features each one being an association of simple features. If the simple features are stored then only some of the more complex ones – only the ones which are used – need to be stored as associations. This is similar to a dictionary storing only some of the infite number of words that may be created from a finite alphabet of letters.

connections between neural assemblies and associative memories. As Santosh Vempala says "*Certainly our work on assemblies can be viewed as closely related to RNNs, with the following remarks*

1. *the nonlinearity is a k-cap: only the top k neurons fire in each round of the RNN.*
2. *the training algorithm is Hebbian plasticity*

And yet, assemblies emerge, with small overlap for distinct stimuli, and allow for pattern completion."

- If deep networks are just a way to associate inputs x_i to outputs y_i with the ability to interpolate among them, there may be simpler way to achieve this goal, without an expensive optimization stage to find the weights W_k. An idea is to combine circular convolution [5] with kernel based dot products.

6 Replacing Backpropagation with layer-wise training and local learning rules

We consider the option to train one layer at the time, keeping fixed all other layers. We start by assuming for simplicity that y_n has the same dimension D as $x_n \quad n = 1, \cdots, D$ and thus for all layers of the network. The process starts with one layer network in which the first layer $W^1_{i,j}$ are randomly initialized with mean zero and unitary Frobenius norm. Then the weights are optimized to minimize the error $\sum ||\ell^1_n||^2 = \sum_n ||\sigma(W^1 x_n) - y_n||^2$; GD is one option for optimization. Then a second layer is added, together with a skip connection from the first layer to the output. The second layer is optimized by minimizing $\sum_n ||\sigma(W^2(\sigma(W^1 x_n)) - \ell^1_n||^2$ over the weights W^2. The process is repeated until the last layer L which gives the oupt of the full network (without skip connections) output $W^L(\cdots(\sigma(W^2(\sigma(W^1 x)))$. At each step one finds the best corrections of the form $\sigma(W^k \cdots (\sigma(W^1 x_n))$ to the previous optimal network. Once the final layer L is added, the first layer is optimized again to minimize the current residual error. Then the first layer is fixed and the second is optimized and so on. Mutatis mutandis the proof of Theorem 4.2 in [18] should apply here showing that the iterative minimization process should converge to a limit estimator which is the optimal L_2 estimator in the class of polynomials with the same degree and monomials as the full network. Notice that minimization at each stage is linear in the weights and can be achieved by gradient descent.

6.1 Invariance in Deep Learning and in Cortex

To simulate the psychophysics experiment of [19], I think a good way is the following:

- run image of the target face i through a deep net pretrained with ImageNet (the face is at a scale within a range of scales); keep last layer activities (before classifier) M_i.

- run image of the test face k (same face or different face at a range of scales); keep activity of last layer T_k.

- train binary classifier on M_i, T_k with correct output reflecting identity (+1 if same, −1 if different) , invariant to scale over a training set of M, T.

The idea is that the classifier will pick up invariant features (for equivariant features t^k and for transformations g_i, it should learn to pick up or compute an invariant $\sum_i \sigma(g_i t^k)$).

Probably good to start with 1) standard pretrained ImageNet network, 2) linear classifier. This can be changed later if necessary. The linear case will train a linear classifier g on top of one or more of the top layers of a pretrained deep net f, where f is a vector of activities. The classifier is given by $g(f(x), f(z)) = A(f(x), f(z))$. The task is to learn A using a training set of images $x_i, i = 1, \cdots, N$ and transformed images Tx_i where T is a legal transformation such as scaling, rotation, affine and their combination. The training set consists of a large ($>> N$) set of input-output examples where the input is $(f(x_i), f(T^k x_m))$ and the output is 1 if $x_m = x_i$ and -1 otherwise.

6.2 Conjecture: Human Robustness to Adversarial Examples

The puzzle is not that adversarial examples exist. Several general underlying mechanisms have been suggested (see for instance [20, 21]), all based on the high dimensionality and overparametrization of the classifiers. The real puzzle is why human vision seems largely immune to the adversarial attacks that affect deep networks. My current conjecture follows from the widespread intuition that human vision disregards perturbations that are not natural or common. For instance Shamir writes "...through a combination of millions of years of evolution and individual experience during the first few years of life (during which the baby is only exposed to natural images), the human visual system had learned to recognize large parts of the manifold of natural images, which includes regions consisting of faces, animals, landscapes, etc. Once the visual system learns this low dimensional manifold, it can perform a projection of any given image to this manifold during its processing." So far, however, this idea does not explain how the visual system learns the manifold of natural images and does not provide a solution to the problem of adversarial attacks. Shamir writes "However, at the moment we do not know how to define this manifold with sufficient precision." My conjecture, if correct, provides an answers. The conjecture says that the manifold of natural images can be characterized by a large set of images – say an ImageNet database – **and** *invariances to several natural transformations:* translation, affine transformations (e.g. scaling, small rotation, stretch etc.), blurring, contrast and illumination changes. Thus a network that can perform recognition invariant to these transformations should also be robust to adversarial attacks. A simple way to check this may be the following. Train a classifier to identify features in the output of a deep net (ResNet pretrained with ImageNet) that are invariant to all transformations for a specific object class (eg faces or cars). If the conjecture is correct the use of these features should avoid adversarial attacks for this class of objects.

We look for a projection matrix P that projects the last layers features onto an invariant manifold. To do this we find P that minimizes $|P(f(x_i) - f(T^k x_m))|^2$ under the constraint

$x_m = x_i$, where f is a vector of activities, $x_i, i = 1, \cdots, N$ and transformed images $T^k x_i$ where T^k is one of the many legal transformations such as scaling, rotation, affine and their combination. . One could use a contrastive loss function that finds Pthat maximize similarity between activities induced by an image and its transforms and maximizes dissimilarity when images are different, irrespectively of transformations. Transformations should be natural transformations for human vision.

6.2.1 In support of the conjecture

Invariances can be imposed via virtual examples (nowdays called data augmentation) or via regularization. As conjectured by [22], regularization and virtual examples can be equivalent. The fact that invariances are equivalent to (many) additional data strongly suggest they may help considerably in defining a manifold of images; the same is true when thinking of invariances corresponding to constraints on the space of functions.

6.3 Conjecture: it is more difficult to learn a compositional target function for a dense network than a CNN (without weight sharing)

The approximation results of [14] find an exponential gap in terms of *rate of approximation* between shallow and CNNs for compositional functions. It is an open problem however whether there is a similar gap in *sample complexity* or *computational complexity* of learning a compositional functions between CNNs and dense networks.

There are results by [23] on sample complexity and by [24] on computational complexity for simple local networks that suggest the conjecture is correct. Both approaches are superficially unrelated to the approximation results. The first approach is based on equivariance properties of networks+algorithms[5].

Lemma 2 *The VC dimension of polynomials in d variables of degree k is $\binom{d-1+k}{d}$, which is the number of monomials of degree at most k formed from d variables.*

Lemma 3 *An upper bound ($m = O(\cdots)$) for any distribution and interpolating algorithms is $m \leq c\frac{VC}{\epsilon} log(\frac{1}{\epsilon}) + \frac{1}{\epsilon} log(\frac{1}{\delta})$.*

A lower bound ($m = \Omega(\cdots)$) on sample complexity is provided by $m \geq c\frac{VC}{\epsilon} + \frac{1}{\epsilon} log\frac{1}{\delta}$ for at least a distribution P_x shattered by $\mathcal{H}$ and any algorithm.

The idea is to use the lower bound for the dense polynomials and the upper bound for the sparse polynomials.

Consider the architecture of the figure with the nonlinearity being $\sigma(z) = z^k$ and pooling on the outputs of the $\frac{d}{2}$ hidden units. For the dense network, the output of each the hidden units is an homogeneous polynomial of degree k in d variables; for the binary tree network each of the $\frac{d}{2}$ hidden units is a polynomial of degree k in 2 variables. The VC dimension of each hidden

[5] My intuition is that equivariance to large groups – like the permutation group – implies large capacity and large sample complexity.

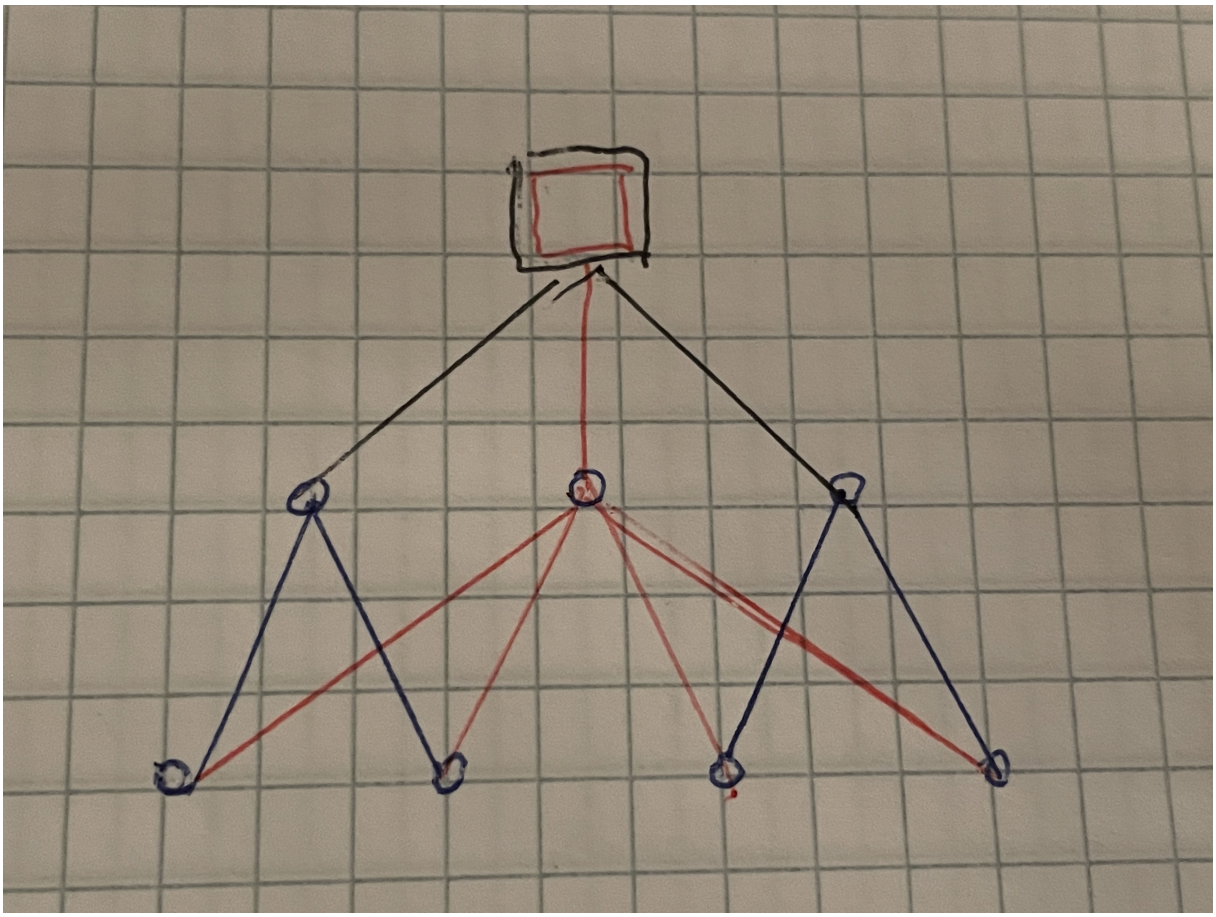


Figure 6: *Assume that at each node there is a univariate polynomial of degree k. The*

unit in the dense network case is $\binom{d+k}{d}$ (which for small k is $\approx d^k$) and for the sparse networks is $\binom{2+k}{2}$ which is $\frac{(k+2)(k+1)}{k}$. For $k=2$ $VC_{dense} = \frac{(d+2)(d+1)}{2}$ and $VC_{sparse} = 6$. Thus the sample complexity of the networks in the Figure is in the order $m_{dense} = O(d^2)$ whereas $m_{sparse} = O(d)$ when in the sparse network there is no weight sharing and $m_{sparse} = O(1)$ with weight sharing. The latter bound agrees with Arora.

Consider the more general case of a dense network somehow computing a full polynomial of degree 2^L in $d = 2^L$ variables: the VC dimension is in the order of d^d. A sparse polynomial of degree 2^L computed by a binary tree with L layers and an activation function $\sigma(z) = z^2$ has a VC dimension proportional to the number of distict monomials which in this case is $\approx 2^L = d$.

It seems to me that it should be possible to use the upper and lower bounds to establish a gap in sample complexity between a homogeneous sparse polynomial computed by a binary tree network vs the homogeneous polynomial computed by a "denser" network. The only missing part is to exhibit a task for which both the dense and the sparse network satisfy the conditions

of the theorem. Such a task for a one layer net could be: detect a local pattern of pixels (2 neighboring white pixels, as suggested by Arora or a k-pattern as suggested by Malach et al.).

The point of this is to illuminate a pretty strong connection between the approximation result – which ultimately depends on number of monomials for the sparse and the shallow networks – and this sample complexity claims. What is missing here are statements about GD.

6.4 Question: learning functions with GD and layerwise minimum-norm least square

It seems that a sparse polynomial can be learned with GD and min norm...question is what is sample complexity for dense and for CNN network? Min norm can be used layerwise...can this work irrespectively of sparse polynomial starting with zero initialization?

6.5 Conjecture: sample complexity for learning hierarchy is big, sample complexity for learning shared weights is small

Can I just use lower bounds on sample complexity based on VC dimension? What is VC dimension of a set of $\frac{d}{2}$ polynomials of degree 2 in 2 variables? It is $\frac{d}{2}6$ whereas VC of polynomial in d variables of degree 2 is $\frac{(d+2)(d+1)}{2}$. Thus the sample complexity is $O(d^k)$ vs $O(d2^k)$. The latter complexity – of the blue network – reduces to $O(2^k)$, that is independent of d for the case of shared weights (because it is enough to train one of the 2-inputs nodes instead of all of them (which are $\frac{d}{2}$).

Let us reformulate VC bounds (given in in approximation paper in terms of number of bits and parameters) as lower bounds in terms of straight VC for polynomial classifiers. Then the estimates above answer the question for simple one hidden layer networks. With polynomial activation functions the analysis can be extended to multilayer nets.

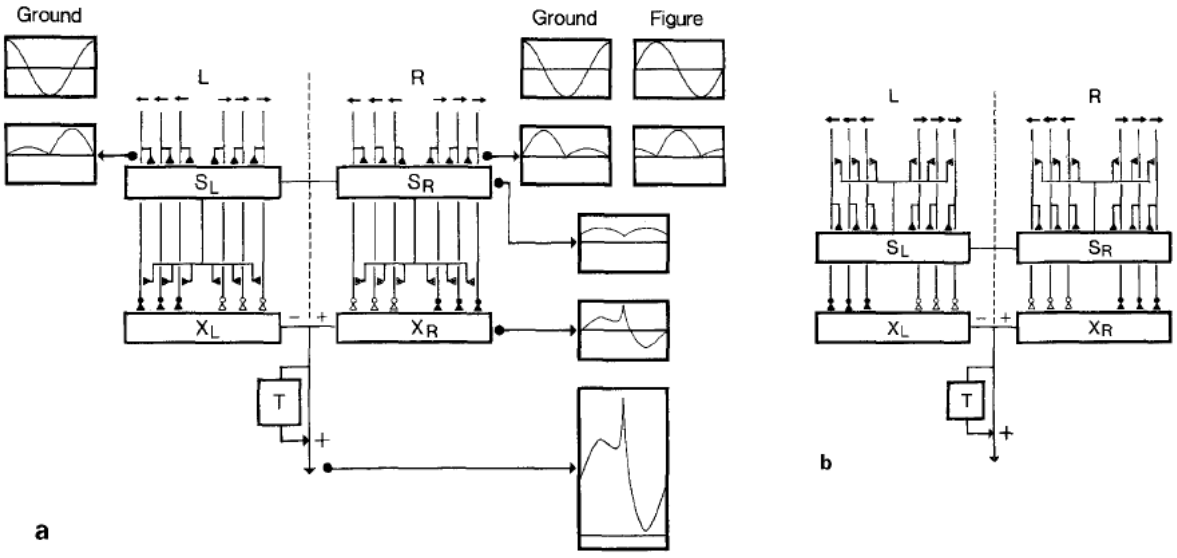


Figure 7: *The neural circuit for figure-ground discrimination in the fly, see [25]*

7 Remarks on neural circuits for transformers

Circuits of the lateral inhibition type are an obvious implementation of the self attention module of transformers. An old version of such nonlinear inhibitory circuits can be found in [25] where the Figure depicts a version of the circuit with a pool cell inhibiting all other neurons (this is equivalent to nonlinear lateral inhibition).