From Associative Memories to Universal Machines

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.

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 Universal Machines

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 into 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 hyperparameters, 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 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
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 $y_i$, $n = 1, \cdots, N$ to a noise-like key vector $x_n$, where $x, y \in \mathbb{R}^D$ and in this simple example $N = D$. The noise-like assumption on the $x_n$ is equivalent to assuming that $X^TX \approx I$, 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 $Ax_i \approx y_i \delta_{i,j} = y_i$. 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

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

(1)

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

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

(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 $D_{k-1}(x_j) = \text{diag}[\sigma'(N_k(x_j))]$ with $N_k(x_j)$ the input to layer $k$. 

3
Radial Basis Function networks

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

implies

\[f(x) = \sum \alpha_i K(x, x_i)\]

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.

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 \(\sigma\) 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. 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 for computational capabilities similar to the Lambda calculus and LISP, along the lines described by Plate in his Holographic Reduced Representations.

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?

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.

\footnote{And the illusions we have about it.}
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.
**Acknowledgments** I am grateful 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


4 Appendix

4.1 Observations about depth

The solution of \( AX = Y \) with \( X \in \mathbb{R}^{D,N} \) is \( A = YX^\dagger = Y(X^TX)^{-1}X^T \). When \( D > N \), \( X^\dagger X = I \) with the matrix \( I \in \mathbb{R}^{N,N} \). 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

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

(3)

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

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

(4)

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 \( D_{k-1}(x_j) = \text{diag}[\sigma'(N_k(x_j))] \) 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 continuous 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

\[
W_{i,j} = K(x_i, x_j) = \sum_\ell \lambda_\ell \phi_\ell(x_i) \Phi_\ell(x_j) = \Phi^T(x_i) \Phi(x_j)
\]

(5)

where the (possibly infinite) column vector \( \Phi \) is \( \Phi(x) = \lambda_\ell^{\frac{1}{2}} \phi_\ell(x) \) and the \( \lambda_\ell \) 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 \( \omega \) drawn from a Gaussian distribution \[6\].

2In 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.
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.

4.2 ResNets

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

\[ W_2 = (I - XX^T), \quad \forall i = 1, \cdots, L - 1, \tag{6} \]

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-expansive 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].

4.3 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 dimensionality.

\[\text{Often this implies some kind of "pooling" (even just by subsampling)}\]
Figure 5: The figure shows the graph of a function of eight variables \((f : \mathbb{R}^8 \rightarrow \mathbb{R})\) 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).

- 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

---

4Starting 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 infinite 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.