Provable Bounds for Learning Some Deep Representations

Sanjeev Arora ARORA@CS.PRINCETON.EDU
Princeton University, Computer Science Department and Center for Computational Intractability, Princeton 08540, USA

Aditya Bhaskara BHASKARA@CS.PRINCETON.EDU
Google Research, New York, NY 10011, USA

Rong Ge RONGGE@MICROSOFT.COM
Microsoft Research, Cambridge, MA 02142, USA

Tengyu Ma TENGYU@CS.PRINCETON.EDU
Princeton University, Computer Science Department and Center for Computational Intractability, Princeton 08540, USA

Abstract

We give algorithms with provable guarantees that learn a class of deep nets in the generative model view popularized by Hinton and others. Our generative model is an n node multilayer network that has degree at most n^γ for some $\gamma < 1$ and each edge has a random edge weight in $[-1, 1]$. Our algorithm learns *almost all* networks in this class with polynomial running time. The sample complexity is quadratic or cubic depending upon the details of the model.

The algorithm uses layerwise learning. It is based upon a novel idea of observing correlations among features and using these to infer the underlying edge structure via a global graph recovery procedure. The analysis of the algorithm reveals interesting structure of neural nets with random edge weights.

1. Introduction

Can we provide theoretical explanation for the practical success of deep nets? Like many other ML tasks, learning deep neural nets is NP-hard, and in fact seems "badly NP-hard" because of many layers of hidden variables connected by nonlinear operations. Usually one imagines that NP-hardness is not a barrier to provable algorithms in ML because the inputs to the learner are drawn from some simple distribution and are not worst-case. This hope was recently borne out in case of generative models such as HMMs, Gaussian Mixtures, LDA etc., for which learning algorithms with provable guarantees were given (Hsu et al., 2012; Moitra & Valiant, 2010; Hsu & Kakade, 2013; Arora et al., 2012; Anandkumar et al., 2012). However, supervised learning of neural nets even on random inputs still seems as hard as cracking cryptographic schemes: this holds for depth-5 neural nets (Jackson et al., 2002) and even ANDs of thresholds (a simple depth two network) (Klivans & Sherstov, 2009).

However, modern deep nets are not " just" neural nets (see the survey (Bengio, 2009)). Instead, the net (or a related net) run *in reverse* from top to bottom is also seen as a a *generative model* for the input distribution. Hinton promoted this viewpoint, and suggested modeling each level as a Restricted Boltzmann Machine (RBM), which is " reversible" in this sense. Vincent et al. (Vincent et al., 2008) suggested modeling each level as a *denoising autoencoder*, consisting of a pair of encoder-decoder functions (see Definition 1). These viewpoints allow *unsupervised, and layerwise learning* in contrast to the supervised learning viewpoint in classical backpropagation. The bottom layer (between observed variables and first layer of hidden variables) is learnt in *unsupervised fashion* using the provided data. This gives values for the first layer of hidden variables, which are used as the data to learn the next layer, and so on. The final net thus learnt is also a good generative model for the distribution of the observed layer. In practice the unsupervised phase is usually used for *pretraining*, and is followed by supervised training[1].

This viewpoint of reversible deep nets is more promising

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

[1] Recent work, e.g., (Krizhevsky et al., 2012) suggests that classical backpropagation does well even without unsupervised pretraining, though the authors suggest that unsupervised pretraining should help further.

for theoretical work and naturally suggests the following problem: *Given samples from a sparsely connected neural network whose each layer is a denoising autoencoder, can the net (and hence its reverse) be learnt in polynomial time with low sample complexity?* Many barriers present themselves in solving such a question. There is no known mathematical condition that describes when a layer in a neural net is a denoising autoencoder. Furthermore, learning even a a *single* layer sparse denoising autoencoder seems much harder than learning sparse-used *overcomplete dictionaries* —which correspond to a single hidden layer with linear operations— and even for this much simpler problem there were no provable bounds at all until the very recent manuscript (Arora et al., 2013)[2].

The current paper presents an interesting family of denoising autoencoders —namely, a sparse network whose weights are randomly chosen in $[-1, 1]$; see Section 2 for details—as well as new algorithms to provably learn almost all models in this family with low time and sample complexity; see Theorem 1. The algorithm outputs a network whose generative behavior is statistically (in ℓ_1 norm) indistinguishable from the ground truth net. (If the weights are discrete, say in $\{-1, 1\}$ then it exactly learns the ground truth net.)

Along the way we exhibit interesting properties of such randomly-generated neural nets. (a) Each pair of adjacent layers constitutes a denoising autoencoder whp see Lemma 2. In other words, the decoder (i.e., sparse neural net with random weights) assumed in the generative model has an associated encoder which is actually the same net run in reverse but with different thresholds at the nodes. (b) The reverse computation is stable to dropouts and noise. (c) The distribution generated by a two-layer net cannot be represented by *any* single layer neural net (see Section 8), which in turn suggests that a random t-layer network cannot be represented by *any* $t/2$-level neural net[3].

Note that properties (a) to (c) are *assumed* in modern deep net work (e.g., reversibility is called "weight tying") whereas they *provably* hold for our random generative model, which is perhaps some theoretical validation of those assumptions.

Context. Recent theoretical papers have analysed multi-level models with hidden features, including SVMs (Cho & Saul, 2009; Livni et al., 2013). However, none solves the task of recovering a ground-truth neural network as we do.

Though real-life neural nets are not random, our consideration of random deep networks makes sense for obtaining theoretical results. Sparse denoising autoencoders are reminiscent of other objects such as error-correcting codes, compressed sensing matrices, etc. which were all first analysed in the random case. As mentioned, provable reconstruction of the hidden layer (i.e., input encoding) in a *known* single layer neural net already seems a nonlinear generalization of compressed sensing, whereas even the usual (linear) version of compressed sensing seems possible only if the adjacency matrix has "random-like" properties (low coherence or restricted isometry or lossless expansion). In fact our result that a single layer of our generative model is a sparse denoising autoencoder can be seen as an analog of the fact that random matrices are good for compressed sensing/sparse reconstruction (see Donoho (Donoho, 2006) for general matrices and Berinde et al. (Berinde et al., 2008) for sparse matrices). Of course, in compressed sensing the matrix of edge weights is known whereas here it has to be learnt, which is the main contribution of our work. Furthermore, we show that our algorithm for learning a single layer of weights can be extended to do layerwise learning of the entire network.

Does our algorithm yield new approaches in practice? We discuss this possibility after sketching our algorithm in the next section.

2. Definitions and Results

Our generative model ("ground truth") has ℓ hidden layers of vectors of binary variables $h^{(\ell)}$, $h^{(\ell-1)}$, .., $h^{(1)}$ (where $h^{(\ell)}$ is the top layer) and an observed layer y (sometimes denoted by $h^{(0)}$) at bottom (see Figure 1). Each layer has n nodes[4]. The weighted graph between layers i and $i+1$ is denoted by $G_i = (U_i, V_i, E_i, W_i)$. Values at U_i correspond to $(h^{(i+1)})$, and those of V_i to $h^{(i)}$, set E_i consists of edges between them and W_i is the weighted adjacency matrix. The degree of this graph is $d = n^\gamma$, and all edge weights are in $[-1, 1]$.

The samples are generated by propagating down the network to the observations. First sample the top layer $h^{(\ell)}$, where the set of nodes that are 1 is picked uniformly among all sets of size $\rho_\ell n$. We call ρ_ℓ the *sparsity* of top level (similarly ρ_i will approximately denote the sparsity of level i). For $i = \ell$ down to 2, each node in layer $i-1$ computes a weighted sum of its neighbors in layer i, and becomes 1 iff that sum strictly exceeds 0. We will use $\mathrm{sgn}(x)$ to denote

[2]The parameter choices in that manuscript make it less interesting in context of deep learning, since the hidden layer is required to have no more than $\sqrt{n}$ nonzeros where n is the size of the observed layer —in other words, the observed vector must be highly compressible.

[3]Formally proving this for $t > 3$ is difficult however since showing limitations of even 2-layer neural nets is a major open problem in computational complexity theory. Some deep learning papers mistakenly cite an old paper for such a result, but the result that actually exists is far weaker.

[4]In the full version of the paper we allow them to have different number of nodes

the threshold function that is 1 if $x > 0$ and 0 else. (Applying $\mathrm{sgn}()$ to a vector involves applying it componentwise.) Thus the network computes as follows: $h^{(i-1)} = \mathrm{sgn}(W_{i-1}h^{(i)})$ for all $i > 1$ and $y = h^{(0)} = W_0 h^{(1)}$ (i.e., no threshold at the observed layer).

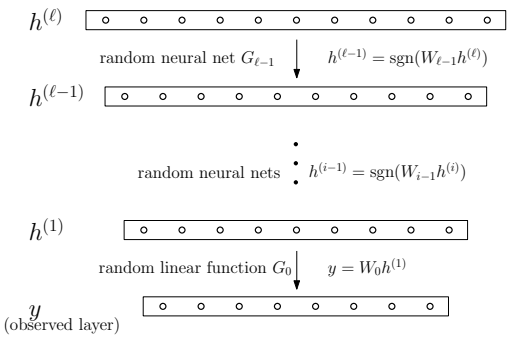

Figure 1. Example of a deep network

Random deep net model: We assume that in the ground truth network, the edges between layers are chosen randomly subject to expected degree d being[5] n^γ, where $\gamma < 1/(\ell+1)$, and each edge $e \in E_i$ carries a weight that is chosen randomly in $[-1, 1]$. This is our model $\mathcal{R}(\ell, \rho_\ell, \{G_i\})$. We also consider —because it leads to a simpler and more efficient learner—a model where edge weights are random in $\{\pm 1\}$ instead of $[-1, 1]$; this is called $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$. Recall that $\rho_\ell > 0$ is such that the $0/1$ vector input at the top layer has 1's in a random subset of $\rho_\ell n$ nodes.

It can be seen that since the network is random of degree d, applying a $\rho_\ell n$-sparse vector at the top layer is likely to produce the following density of 1's (approximately) at the successive layers: $\rho_\ell d/2, \rho_\ell (d/2)^2$, etc.. We assume the density of last layer $\rho_\ell d^\ell / 2^\ell = O(1)$. This way the density at the last-but-one layer is $o(1)$, and the last layer is real-valued and dense.

Now we state our main result. Note that $1/\rho_\ell$ is at most n.

Theorem 1 *When degree $d = n^\gamma$ for $0 < \gamma \leq 0.2$, density $\rho_\ell (d/2)^\ell = C$ for some large constant C[6], the network model $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$ can be learnt[7] using $O(\log n/\rho_\ell^2)$ samples and $O(n^2 \ell)$ time. The network model $\mathcal{R}(\ell, \rho_\ell, \{G_i\})$ can be learnt in polynomial time and using $O(n^3 \ell^2 \log n/\eta^2)$ samples* [8].

[5] In the full version we allow degrees to be different for different layers.

[6] Notice that this number is roughly the expected number of features that are on and influence a particular observed variable. When C is a large constant the output has sparsity close to 1.

[7] The algorithm outputs a network that is equivalent to the ground truth up to permutation of hidden variables

[8] Here the learning algorithm outputs a network; the distribution generated by this output for all but the last layer is within total variation distance η to the true distribution.

Algorithmic ideas. We are unable to analyse existing algorithms. Instead, we give new learning algorithms that exploit the very same structure that makes these random networks interesting in the first place i.e., each layer is a denoising autoencoder. The crux of the algorithm is a new twist on the old Hebbian rule (Hebb, 1949) that "Things that fire together wire together." In the setting of layerwise learning, this is adapted as follows: "Nodes in the same layer that fire together a lot are likely to be connected (with positive weight) to the same node at the higher layer." The algorithm consists of looking for such pairwise (or 3-wise) correlations and putting together this information globally. The global procedure boils down to the graph-theoretic problem of reconstructing a bipartite graph given pairs of nodes that share a common neighbor (see Section 6). This is a variant of the GRAPH SQUARE ROOT problem which is NP-complete on worst-case instances but solvable for sparse random (or random-like) graphs.

Note that existing neural net algorithms (to the extent that they are Hebbian) can also be seen as leveraging correlations. But putting together this information is done via the language of nonconvex optimization (i.e., an objective function with suitable penalty terms). Our ground truth network is indeed a particular local optimum in any reasonable formulation. It would be interesting to show that existing algorithms provably find the ground truth in polynomial time but currently this seems difficult.

Can our new ideas be useful in practice? We think that using a global reconstruction procedure that leverages local correlations seems promising, especially if it avoids nonconvex optimization. Our proof currently needs that the hidden layers are sparse, and the edge structure of the ground truth network is "random like" (in the sense that two distinct features at a level tend to affect *fairly* disjoint sets of features at the next level). We need this assumption both for inferring correlations and the reconstruction step.

Finally, note that random neural nets do seem useful in so-called *reservoir* computing, and more structured neural nets with random edge weights are also interesting in feature learning (Saxe et al., 2011). So perhaps they do provide useful representational power on real data. Such empirical study is left for future work.

Throughout, we need well-known properties of random graphs with expected degree d, such as the fact that they are expanders (Hoory et al., 2006); these properties appear in the supplementary material. The most important one, unique neighbors property, appears in the next Section.

3. Each layer is a Denoising Auto-encoder

As mentioned earlier, modern deep nets research often assumes that the net (or at least some layers in it) should

approximately preserve information, and even allows easy going back/forth between representations in two adjacent layers (what we earlier called "reversibility"). Below, y denotes the lower layer and h the higher (hidden) layer. Popular choices of s include logistic function, soft max, etc.; we use a simple threshold function in our model.

Definition 1 (DENOISING AUTOENCODER) *An* autoencoder *consists of a* decoding function $D(h) = s(Wh + b)$ *and an* encoding function $E(y) = s(W'y + b')$ *where W, W' are linear transformations, b, b' are fixed vectors and s is a nonlinear function that acts identically on each coordinate. The autoencoder is* denoising *if $E(D(h)+\xi) = h$ with high probability where h is drawn from the distribution of the hidden layer, ξ is a noise vector drawn from the noise distribution, and $D(h) + \xi$ is a shorthand for "$D(h)$ corrupted with noise ξ." The autoencoder is said to use* weight tying *if $W' = W^T$.*

In empirical work the denoising autoencoder property is only implicitly imposed on the deep net by minimizing the reconstruction error $||y - D(E(y + \xi))||$, where ξ is the noise vector. Our definition is slightly different but is actually stronger since y is exactly $D(h)$ according to the generative model. Our definition implies the existence of an encoder E that makes the penalty term exactly zero. We show that in our ground truth net (whether from model $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$ or $\mathcal{R}(\ell, \rho_\ell, \{G_i\})$) every graph G_i whp satisfies this definition, and with weight-tying.

We show a single-layer random network is a denoising autoencoder if the input layer is a random ρn sparse vector, and the output layer has density $\rho d/2 < 1/20$.

Lemma 2 *If $\rho d < 0.1$ (i.e., the bottom layer is also fairly sparse) then the single-layer network $G(U, V, E, W)$ in $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$ or $\mathcal{R}(\ell, \rho_\ell, \{G_i\})$ is a denoising autoencoder with high probability (over the choice of the random graph and weights), where the noise flips every output bit independently with probability 0.1. It uses weight tying.*

The proof of this lemma highly relies on a property of random graph, called the *strong unique-neighbor property.*

For any node $u \in U$, let $F(u)$ be its neighbors in V; for any subset $S \subset U$, let $UF(u, S)$ be the set of *unique* neighbors of u with respect to S, i.e.,

$$UF(u, S) \triangleq \{v \in V : v \in F(u), v \notin F(S \setminus \{u\})\}$$

Property 1 *In a bipartite graph $G(U, V, E, W)$, a node $u \in U$ has $(1 - \epsilon)$-unique neighbor property (UNP) with respect to S if*

$$\sum_{v \in UF(u,S)} |W(u, v)| \geq (1 - \epsilon) \sum_{v \in F(u)} |W(u, v)| \quad (1)$$

The set S has $(1 - \epsilon)$-strong UNP if every node in U has $(1 - \epsilon)$-UNP with respect to S.

Note that we don't need this property to hold for all sets of size ρn (which is possible if ρd is much smaller). However, for any fixed set S of size ρn, this property holds with high probability over the randomness of the graph.

Now we sketch the proof for Lemma 2 (details are in full version) when the edge weights are in $\{-1, 1\}$.

First, the decoder definition is implicit in our generative model: $y = \text{sgn}(Wh)$. (That is, $b = \vec{0}$ in the autoencoder definition.) Let the encoder be $E(y) = \text{sgn}(W^T y + b')$ for $b' = 0.2d \times \vec{1}$.In other words, the same bipartite graph and different thresholds can transform an assignment on the lower level to the one at the higher level.

To prove this consider the strong unique-neighbor property of the network. For the set of nodes that are 1 at the higher level, a majority of their neighbors at the lower level are unique neighbors.The unique neighbors with positive edges will always be 1 because there are no -1 edges that can cancel the $+1$ edge (similarly the unique neighbors with negative edges will always be 0). Thus by looking at the set of nodes that are 1 at the lower level, one can infer the correct $0/1$ assignment to the higher level by doing a simple threshold of say $0.2d$ at each node in the higher layer.

4. Learning a single layer network

Our algorithm, outlined below (Algorithm 1), learns the network layer by layer starting from the bottom. We now focus on learning a single-layer network, which as we noted amounts to learning *nonlinear* dictionaries with random dictionary elements. The algorithm illustrates how we leverage the sparsity and the randomness of the edges, and use pairwise or 3-wise correlations combined with RecoverGraph procedure of Section 6.

Algorithm 1 High Level Algorithm

Input: samples y's generated by a deep network described in Section 2

Output: the network/encoder and decoder functions

1: **for** $i = 1$ TO l **do**
2: Construct *correlation graph* using samples of $h^{(i-1)}$
3: Call RecoverGraph to learn the positive edges E_i^+
4: Use PartialEncoder to encode all $h^{(i-1)}$ to $h^{(i)}$
5: Use LearnGraph/LearnDecoder to learn the graph/decoder between layer i and $i - 1$.
6: **end for**

For simplicity we describe the algorithm when edge weights are $\{-1, 1\}$, and sketch the differences for real-valued weights at the end of this section.

As before we use ρ to denote the sparsity of the hidden layer. Say two nodes in the observed layer are *related* if they have a common neighbor in the hidden layer to which they are attached via a $+1$ edge. The algorithm first tries to identify all related nodes, then use this information to learn all the positive edges. With positive edges we show it is already possible to recover the hidden variables. Finally given both hidden variables and observed variables the algorithm finds all negative edges.

STEP 1: *Construct correlation graph:* This step uses a new twist on the Hebbian rule to identify correlated nodes.[9]

Algorithm 2 PairwiseGraph

Input: $N = O(\log n/\rho)$ samples of $y = \mathrm{sgn}(Wh)$,
Output: $\hat{G}$ on vertices V, u, v connected if related
  for each u, v in the output layer **do**
    if $\geq \rho N/3$ samples have $y_u = y_v = 1$ **then**
      connect u and v in $\hat{G}$
    end if
  end for

Claim *In a random sample of the output layer, related pairs u, v are both 1 with probability at least 0.9ρ, while unrelated pairs are both 1 with probability at most $(\rho d)^2$.*

(Proof Sketch): First consider a related pair u, v, and let z be a vertex with $+1$ edges to u, v. Let S be the set of neighbors of u, v excluding z. The size of S cannot be much larger than $2d$. Under the choice of parameters, we know $\rho d \ll 1$, so the event $h_S = \vec{0}$ conditioned on $h_z = 1$ has probability at least 0.9. Hence the probability of u and v being both 1 is at least 0.9ρ. Conversely, if u, v are unrelated then for both u, v to be 1 there must be nodes y and z such that $h_y = h_z = 1$, and are connected to u and v respectively via $+1$ edges. The probability of this event is at most $(\rho d)^2$ by union bound.

Thus, if $(\rho d)^2 < 0.1\rho$, using $O(\log n/\rho^2)$ samples we can estimate the correlation of all pairs accurately enough, and hence recover all related pairs whp.

STEP 2: *Use RecoverGraph procedure to find all edges that have weight $+1$.* (See Section 6 for details.)

STEP 3: *Using the $+1$ edges to encode all the samples y.*

Algorithm 3 PartialEncoder

Input: positive edges E^+, $y = \mathrm{sgn}(Wh)$, threshold θ
Output: the hidden variable h
  Let M be the indicator matrix of E^+ ($M_{i,j} = 1$ iff $(i, j) \in E^+$)
  return $h = \mathrm{sgn}(M^T y - \theta\vec{1})$

[9]The lowermost (real valued) layer is treated slightly differently – see Section 7.

Although we have only recovered the positive edges, we can use PARTIALENCODER algorithm to get h given y!

Lemma 3 *If support of h satisfies $11/12$-strong unique neighbor property, and $y = \mathrm{sgn}(Wh)$, then Algorithm 3 outputs h with $\theta = 0.3d$.*

This uses the unique neighbor property: for every z with $h_z = 1$, it has at least $0.4d$ unique neighbors that are connected with $+1$ edges. All these neighbors must be 1 so $[(E^+)^T y]_z \geq 0.4d$. On the other hand, for any z with $h_z = 0$, the unique neighbor property implies that z can have at most $0.2d$ positive edges to the $+1$'s in h. Hence $h = \mathrm{sgn}((E^+)^T y - 0.3d\vec{1})$.

STEP 4: *Recover all weight -1 edges.*

Algorithm 4 Learning Graph

Input: positive edges E^+, samples of (h, y)
Output: E^-
1: $R \leftarrow (U \times V) \setminus E^+$.
2: **for** each of the samples (h, y), and each v **do**
3: Let S be the support of h
4: **if** $y_v = 1$ and $S \cap B^+(v) = \{u\}$ for some u **then**
5: For all $s \in S$ remove (s, v) from R
6: **end if**
7: **end for**
8: **return** R

Now consider many pairs of (h, y), where h is found using Step 3. Suppose in some sample, $y_u = 1$ for some u, and exactly one neighbor of u in the $+1$ edge graph (which we know entirely) is in $\mathrm{supp}(h)$. Then we can conclude that for any z with $h_z = 1$, there cannot be a -1 edge (z, u), as this would cancel out the unique $+1$ contribution.

Lemma 4 *Given $O(\log n/(\rho^2 d))$ samples of pairs (h, y), with high probability (over the random graph and the samples) Algorithm 4 outputs the correct set E^-.*

To prove this lemma, we just need to bound the probability of the following events for any non-edge (x, u): $h_x = 1$, $|\mathrm{supp}(h) \cap B^+(u)| = 1$, $\mathrm{supp}(h) \cap B^-(u) = \emptyset$ (B^+, B^- are positive and negative parents). These three events are almost independent, the first has probability ρ, second has probability $\approx \rho d$ and the third has probability almost 1.

Leveraging 3-wise correlation: The above sketch used pairwise correlations to recover the $+1$ weights when $\rho \ll 1/d^2$. It turns out that using 3-wise correlations allow us to find correlations under a weaker requirement $\rho < 1/d^{3/2}$. Now call three observed nodes u, v, s *related* if they are connected to a common node at the hidden layer via $+1$ edges. Then we can prove a claim analogous to the one above, which says that for a related triple, the probability

that u, v, s are all 1 is at least 0.9ρ, while the probability for unrelated triples is roughly at most $(\rho d)^3$. Thus as long as $\rho < 0.1/d^{3/2}$, it is possible to find related triples correctly. The RecoverGraph algorithm can be modified to run on 3-uniform hypergraph consisting of these related triples to recover the $+1$ edges.

The end result is the following theorem. This is the algorithm used to get the bounds stated in our main theorem.

Theorem 5 *Suppose our generative neural net model with weights $\{-1, 1\}$ has a single layer and the assignment of the hidden layer is a random ρn-sparse vector, with $\rho \ll 1/d^{3/2}$. Then there is an algorithm that runs in $O(n(d^3 + n))$ time and uses $O(\log n/\rho^2)$ samples to recover the ground truth with high probability over the randomness of the graph and the samples.*

When weights are real numbers. We only sketch this and leave the details to the full version. Surprisingly, steps 1, 2 and 3 still work. In the proofs, we have only used the sign of the edge weights – the magnitude of the edge weights can be arbitrary. This is because the proofs in these steps relies on the unique neighbor property. If some node is on (has value 1), then its unique positive neighbors at the next level will always be on, no matter how small the positive weights might be. Also notice in PartialEncoder we are only using the support of E^+, but not the weights.

After Step 3 we have turned the problem of unsupervised learning to a supervised one in which the outputs are just linear classifiers over the inputs! Thus the weights on the edges can be learnt to any desired accuracy.

5. Correlations in a Multilayer Network

We now consider multi-layer networks, and show how they can be learnt layerwise using a slight modification of our one-layer algorithm at each layer. At a technical level, the difficulty in the analysis is the following: in single-layer learning, we assumed that the higher layer's assignment is a random ρn-sparse binary vector, however in the multilayer network, the assignments in intermediate layers do not satisfy this. We will show nonetheless that the the correlations at intermediate layers are low enough, so our algorithm can still work. Again for simplicity we describe the algorithm for the model $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$, in which the edge weights are ± 1. Recall that ρ_i's are the "expected" number of 1s in the layer $h^{(i)}$. Because of the unique neighbor property, we expect roughly $\rho_\ell(d/2)$ fraction of $h^{(\ell-1)}$ to be 1. Hence $\rho_i = \rho_\ell \cdot (d/2)^{\ell-i}$.

Lemma 6 *Consider a network from $\mathcal{D}(\ell, \rho_\ell, \{G_i\})$. With high probability (over the random graphs between layers) for any two nodes u, v in layer $h^{(1)}$,*

$$\Pr[h_u^{(1)} = h_v^{(1)} = 1] \begin{cases} \geq \rho_2/2 & \text{if } u, v \text{ related} \\ \leq \rho_2/4 & \text{otherwise} \end{cases}$$

(Proof Sketch): Call a node s at the topmost layer an ancestor of u if there is a path of length $\ell - 1$ from s to u. The number of ancestors of a node u is roughly $2^{\ell-1}\rho_1/\rho_\ell$. With good probability there is at most one ancestor of u that has value 1.Call s a positive ancestor of u, if when only s is 1 at the topmost layer, the node u is 1. The number of positive ancestors of u is roughly ρ_1/ρ_ℓ.

The probability that a node u is on is roughly proportional to the number of positive ancestors. For a pair of nodes, if they are both 1, either one of their common positive ancestor is 1, or both of them have a positive ancestor that is 1. The probability of the latter is $O(\rho_1^2)$ which by assumption is much smaller than ρ_2.

When u, v share a common positive parent z, the positive ancestors of z are all common positive ancestors of u, v (so there are at least ρ_2/ρ_ℓ). The probability that one of positive common ancestor is on and no other ancestors are on is at least $\rho_2/2$.

If u, v do not share a common parent, then the number of their common positive ancestors depends on how many "common grandparents" (common neighbors in layer 3) they have. We show with high probability (over the graph structure) the number of common positive ancestors is small. Therefore the probability that both u,v are 1 is small. □

Once we can identify related pairs by Lemma 6, the Steps 2,3,4 in the single layer algorithm still work. We can learn the bottom layer graph and get $h^{(2)}$. This argument can be repeated after 'peeling off' the bottom layer, thus allowing us to learn layer by layer.

6. Graph Recovery

Graph reconstruction consists of recovering a graph given information about its subgraphs (Bondy & Hemminger, 1977). A prototypical problem is the *Graph Square Root* problem, which calls for recovering a graph given all pairs of nodes whose distance is at most 2. This is NP-hard.

Definition 2 (Graph Recovery) *Let $G_1(U, V, E_1)$ be an unknown random bipartite graph between $|U| = n$ and $|V| = n$ nodes where each edge is picked with probability d/n independently.*

Given: *Graph $G(V, E)$ where $(v_1, v_2) \in E$ iff v_1 and v_2 share a common parent in G_1 (i.e. $\exists u \in U$ where $(u, v_1) \in E_1$ and $(u, v_2) \in E_1$).*

Goal: *Find the bipartite graph G_1.*

Some of our algorithms (using 3-wise correlations) need to solve analogous problem where we are given triples of nodes which are mutually at distance 2 from each other, which we will not detail for lack of space.

We let $F(S)$ (resp. $B(S)$) denote the set of neighbors of $S \subseteq U$ (resp. $\subseteq V$) in G_1. Also $\Gamma(\cdot)$ gives the set of neighbors in G. Now for the recovery algorithm to work, we need the following properties (all satisfied whp by random graph when $d^3/n \ll 1$):

1. For any $v_1, v_2 \in V$, $|(\Gamma(v_1) \cap \Gamma(v_2)) \backslash (F(B(v_1) \cap B(v_2)))| < d/20$.
2. For any $u_1, u_2 \in U$, $|F(u_1) \cup F(u_2)| > 1.5d$.
3. For any $u \in U$, $v \in V$ and $v \notin F(u)$, $|\Gamma(v) \cap F(u)| < d/20$.
4. For any $u \in U$, at least 0.1 fraction of pairs $v_1, v_2 \in F(u)$ does not have a common neighbor other than u.

The first property says "most correlations are generated by common cause": all but possibly $d/20$ of the common neighbors of v_1 and v_2 in G, are in fact neighbors of a common neighbor of v_1 and v_2 in G_1.

The second property says the sets $F(u)$ should not overlap much. This is clear because the sets are random.

The third property says if a node v is not 'caused by' u, then it is not correlated with too many neighbors of u.

The fourth property says every cause introduces a significant number of correlations that are unique to that cause.

In fact, Properties 2-4 are closely related from the unique neighbor property.

Lemma 7 *When graph G_1 satisfies Properties 1-4, Algorithm 5 successfully recovers G_1 in expected time $O(n^2)$.*

(Proof Sketch): We first show that when (v_1, v_2) has more than one unique common cause, then the condition in the if statement must be false. This follows from Property 2. We know the set S contains $F(B(v_1) \cap B(v_2))$. If $|B(v_1) \cap B(v_2)| \geq 2$ then Property 2 says $|S| \geq 1.5d$, which implies the condition in the if statement is false.

Then we show if (v_1, v_2) has a unique common cause u, then S' will be equal to $F(u)$. By Property 1, we know $S = F(u) \cup T$ where $|T| \leq d/20$. Now, every v in $F(u)$ is connected to every other node in $F(u)$. Therefore $|\Gamma(v) \cap S| \geq |\Gamma(v) \cap F(u)| \geq 0.8d - 1$, and $v \in S'$.

For any node v' outside $F(u)$, by Property 3 it can only be connected to $d/20$ nodes in $F(u)$. Therefore $|\Gamma(v) \cap S| \leq |\Gamma(v) \cap F(u)| + |T| \leq d/10$. Hence v' is not in S'. Following these arguments, S' must be equal to $F(u)$, and the algorithm successfully learns the edges related to u.

The algorithm will successfully find all nodes $u \in U$ because of Property 4: for every u there are enough number of edges in G that is only caused by u. When one of them is sampled, the algorithm successfully learns the node u.

Finally we bound the running time. By Property 4 we know that the algorithm identifies a new node $u \in U$ in at most 10 iterations in expectation. Each iteration takes at most $O(n)$ time. Therefore the algorithm takes at most $O(n^2)$ time in expectation. □

Algorithm 5 RecoverGraph

Input: G given as in Definition 2
Output: Find the graph G_1 as in Definition 2.
repeat
  Pick a random edge $(v_1, v_2) \in E$.
  Let $S = \{v : (v, v_1), (v, v_2) \in E\}$.
  if $|S| < 1.3d$ **then**
    $S' = \{v \in S : |\Gamma(v) \cap S| \geq 0.8d - 1\}$ {S' should be a clique in G}
    In G_1, create node u, connect u to every $v \in S'$.
    Mark all the edges (v_1, v_2) for $v_1, v_2 \in S'$.
  end if
until all edges are marked

7. Learning the lowermost (real-valued) layer

Note that in our model, the lowest (observed) layer is real-valued and does not have threshold gates. Thus our earlier learning algorithm cannot be applied as is. However, we see that the same paradigm – identifying correlations and using RecoverGraph – can be used.

The first step is to show that for a random weighted graph G, the linear decoder $D(h) = Wh$ and the encoder $E(y) = \mathrm{sgn}(W^T y + b)$ form a denoising autoencoder with real-valued outputs, as in Bengio et al. (Bengio et al., 2013).

Lemma 8 *If G is a random weighted graph, the encoder $E(y) = \mathrm{sgn}(W^T y - 0.4d\vec{1})$ and linear decoder $D(h) = Wh$ form a denoising autoencoder, for noise vectors γ which have independent components, each having variance at most $O(d/\log^2 n)$.*

The next step is to show a bound on correlations as before. For simplicity we state it assuming the layer $h^{(1)}$ has a random $0/1$ assignment of sparsity ρ_1. In the full version we consider the correlations introduced by higher layers as we did in Section 5.

Theorem 9 *When $\rho_1 d = O(1)$, $d = \Omega(\log^2 n)$, with high probability over the choice of the weights and the choice of the graph, for any three nodes u, v, s the assignment y to the bottom layer satisfies:*

1. *If u, v and s have no common neighbor, then $|\mathbb{E}_h[y_u y_v y_s]| \leq \rho_1/3$*

2. *If u, v and s have a unique common neighbor, then* $|\mathbb{E}_h[y_u y_v y_s]| \geq 2\rho_1/3$

8. Two layers are more expressive than one

In this section we show that a two-layer network with ± 1 weights is more expressive than one layer network with arbitrary weights. A two-layer network (G_1, G_2) consists of random graphs G_1 and G_2 with random ± 1 weights on the edges. Viewed as a generative model, its input is $h^{(3)}$ and the output is $h^{(1)} = \mathrm{sgn}(W_1 \,\mathrm{sgn}(W_2 h^{(3)}))$. We will show that a single-layer network even with arbitrary weights and arbitrary threshold functions must generate a fairly different distribution.

Lemma 10 *For almost all choices of (G_1, G_2), the following is true. For every one layer network with matrix A and vector b, if $h^{(3)}$ is chosen to be a random $\rho_3 n$-sparse vector with $\rho_3 d_2 d_1 \ll 1$, the probability (over the choice of $h^{(3)}$) is at least $\Omega(\rho_3^2)$ that* $\mathrm{sgn}(W_1 \,\mathrm{sgn}(W_2 h^{(3)})) \neq \mathrm{sgn}(Ah^{(3)} + b)$.

The idea is that the cancellations possible in the two-layer network simply cannot all be accomodated in a single-layer network even using arbitrary weights. More precisely, even the bit at a single output node v cannot be well-represented by a simple threshold function.

First, observe that the output at v is determined by values of $d_1 d_2$ nodes at the top layer that are its ancestors. It is not hard to show in the one layer net (A, b), there should be no edge between v and any node u that is not its ancestor. Then consider structure in Figure 2. Assuming all other parents of v are 0 (which happens with probability at least 0.9), and focus on the values of (u_1, u_2, u_3, u_4). When these values are $(1, 1, 0, 0)$ and $(0, 0, 1, 1)$, v is off. When these values are $(1, 0, 0, 1)$ and $(0, 1, 1, 0)$, v is on. This is impossible for a one layer network because the first two ask for $\sum_{A_{u_i,v}} +2b_v \leq 0$ and the second two ask for $\sum_{A_{u_i,v}} +2b_v < 0$.

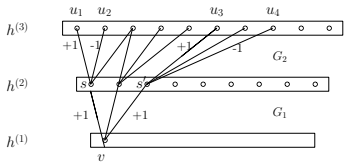

Figure 2. Two-layer network(G_1, G_2)

9. Experiments

We did a basic implementation of our algorithms and ran experiments on synthetic data. The results show that our analysis has the right asymptotics, and that the constants involved are all small. As is to be expected, the most expensive step is that of finding the negative edges (since the algorithm is to 'eliminate non-edges'). However, stopping the algorithm after a certain number of iterations gives a reasonable over-estimate for the negative edges.

The following are the details of one experiment: a network with two hidden layers and one observed layer ($\ell = 2$, both layers use thresholds) with $n = 5000$, $d = 16$ and $\rho_\ell n = 4$; the weights are random $\{\pm 1\}$ but we made sure that each node has exactly 8 positive edges and 8 negative edges. Using $500,000$ samples, in less than one hour (on a single 2GHz processor) the algorithm correctly learned the first layer, and the positive edges of the second layer. Learning the negative edges of second layer (as per our analysis) requires many more samples. However using 5×10^6 samples, the algorithm makes only 10 errors in learning negative edges.

10. Conclusions

Many aspects of deep nets are mysterious to theory: reversibility, why use denoising autoencoders, why this highly non-convex problem is solvable, etc. Our paper gives a first-cut explanation. Worst-case nets seem hard, and rigorous analysis of interesting subcases can stimulate further development: see e.g., the role played in Bayes nets by rigorous analysis of message-passing on trees and graphs of low tree-width. We chose to study randomly generated nets, which makes scientific sense (nonlinear generalization of random error correcting codes, compressed sensing etc.), and also has some empirical support, e.g. in reservoir computing.

The very concept of a denoising autoencoder (with weight tying) suggests to us a graph with random-like properties. We would be very interested in an empirical study of the randomness properties of actual deep nets learnt via supervised backprop. (For example, in (Krizhevsky et al., 2012) the lower layers use convolution, which is decidedly non-random. But higher layers are learnt by backpropagation initialized with a complete graph and may end up more random-like.)

Network randomness is not so crucial in our analysis of learning a single layer, but crucial for layerwise learning: the randomness of the graph structure is crucial for controlling (i.e., upper bounding) correlations among features appearing in the same hidden layer (see Lemma 6). Provable layerwise learning under weaker assumptions would be very interesting.

Acknowledgements

We would like to thank Yann LeCun, Ankur Moitra, Sushant Sachdeva, Linpeng Tang for numerous helpful discussions throughout various stages of this work and anonymous reviewers for their helpful feedback and comments.

References

Anandkumar, Anima, Foster, Dean P., Hsu, Daniel, Kakade, Sham M., and Liu, Yi-Kai. A spectral algorithm for latent Dirichlet allocation. In *Advances in Neural Information Processing Systems 25*, 2012.

Arora, Sanjeev., Ge, Rong., and Moitra, Ankur. Learning topic models – going beyond svd. In *IEEE 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick NJ, USA, October 20-23*, pp. 1–10, 2012.

Arora, Sanjeev, Ge, Rong, and Moitra, Ankur. New algorithms for learning incoherent and overcomplete dictionaries. *ArXiv*, 1308.6273, 2013.

Bengio, Yoshua. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

Bengio, Yoshua, Courville, Aaron C., and Vincent, Pascal. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35 (8):1798–1828, 2013.

Berinde, R., Gilbert, A.C., Indyk, P., Karloff, H., and Strauss, M.J. Combining geometry and combinatorics: a unified approach to sparse signal recovery. In *46th Annual Allerton Conference on Communication, Control, and Computing*, pp. 798–805, 2008.

Bondy, J Adrian and Hemminger, Robert L. Graph reconstructiona survey. *Journal of Graph Theory*, 1(3):227–268, 1977.

Cho, Youngmin and Saul, Lawrence. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems 22*, pp. 342–350. 2009.

Donoho, David L. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

Hebb, Donald O. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, new edition edition, June 1949.

Hoory, Shlomo, Linial, Nathan, and Wigderson, Avi. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

Hsu, Daniel and Kakade, Sham M. Learning mixtures of spherical gaussians: moment methods and spectral decompositions. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pp. 11–20, 2013.

Hsu, Daniel, Kakade, Sham M., and Zhang, Tong. A spectral algorithm for learning hidden Markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Jackson, Jeffrey C, Klivans, Adam R, and Servedio, Rocco A. Learnability beyond ac^0. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 776–784. ACM, 2002.

Klivans, Adam R and Sherstov, Alexander A. Cryptographic hardness for learning intersections of halfspaces. *Journal of Computer and System Sciences*, 75(1):2–12, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoff. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114. 2012.

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. A provably efficient algorithm for training deep networks. *ArXiv*, 1304.7045, 2013.

Moitra, Ankur and Valiant, Gregory. Settling the polynomial learnability of mixtures of gaussians. In *the 51st Annual Symposium on the Foundations of Computer Science (FOCS)*, 2010.

Saxe, Andrew, Koh, Pang W, Chen, Zhenghao, Bhand, Maneesh, Suresh, Bipin, and Ng, Andrew. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1089–1096, 2011.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *ICML*, pp. 1096–1103, 2008.