Review of Provable Bounds for Learning Some Deep Representations [1]

Sasank Chilamkurthy

October 22, 2014

Abstract

In [1], Authors give provable bounds for a algorithm to learn a interesting family of deep neural networks using properties of Random graphs and leveraging sparsity assumptions of this family. We summarize [1] and explain its main contributions.

1 Introduction

Deep Neural Nets have been very successful in tasks like vision, speech processing etc. Like many ML tasks, learning deep nets is NP-hard and need nonlinear optimization of many variables. Usually one imagines that this is not really a barrier to provable algorithms as inputs to learner are drawn from simple distributions and are not worst case. Provable guarantees for learning were given recently in case of generative models like HMMs, Gaussian Mixtures, LDA etc. [2–6] However, supervised learning of neural nets even on random inputs still seems as hard as cryptographic schemes. [7, 8]

However, modern deep nets are not "just" neural nets. (see [10]) Instead the net is also seen as generative model for as generative model for input distribution when run in reverse. Hinton promoted this viewpoint and suggested modelling each level as a Restricted Boltzmann Machine (RBM), which is "reversible" in this sense. Vincent et al. [9] suggested modelling each layer as a denoising autoencoder, a generalization of the RBM.

These viewpoints allow a unsupervised and layerwise learning methodology instead of classical backpropagation. The bottom layer is learnt in unsupervised fashion using the provided data. This gives values for the next layer of hidden variables, which are used as the data to learn the next higher layer, and so on. The final net thus learnt is also a good generative model for the distribution of the bottom layer. In practice the unsupervised phase is followed by supervised training.

The current paper present both an interesting family of denoising autoencoders – a sparse neural network whose weights are randomly chosen in $[-1, 1]$ – as well as new algorithms to provably learn almost all models in this family.

Though real-life neural nets are not random, consideration of random deep networks makes some sense for theory. Sparse denoising autoencoders are reminiscent of other objects such as error-correcting codes, compressed sensing, etc. which were all first analysed in the random case.

Here's the outline of our summary: First, generative model is introduced and a highlevel algorithm is outlined. Next, each layer in the model is shown to be a denoising autoencoder. Then, Algorithms are described and analyzed for single layer. Finally, extensions are made to analyse multilayer networks

2 Definitions and Results

Let's introduce some notation and list the assumptions on generative model. Then main results are introduced.

2.1 Generative Model

Generative neural net model has l hidden layers of binary vectors $h^{(l)}, h^{(l-1)}, \ldots, h^{(1)}$ and an observed layer y at bottom. The number of vertices at layer i is n[1]and set of edges between layers $h^{(i)}$ and $h^{(i+1)}$ by E_i. Weighted graph between layers $h^{(i)}$ and $h^{(i+1)}$ has degree at most n^γ and all edge weights are in $[-1, 1]$. The generative model works like a neural net with threshold 0 at all nodes expect last layer. i.e, $h^{(i-1)} = \text{sgn}(G_{i-1}h^{(i)})$ for all $i > 0$ and $h^{(0)} = G_0 h^{(1)}$ where G_i is weight matrix of bipartite graph between layers i and $i+1$. The top layer $h^{(l)}$ is initialized with a 0/1 assignment where the set of nodes that are 1 is picked uniformly among all sets of size $\rho_l n$.

The set of forward neighbours of u in graph G_i is denoted by $F_i(u) = \{v : (u, v) \in E_i\}$ and set of backward neighbours of v in G_i is denoted by $B_i(v) = \{u : (u, v) \in E_i\}$. $F_i^+(u) = \{v : (u, v) \in E_i^+\}$ is set of positive forward neighbours of u and $B_i^+(v)$ is set of positive backward neighbours of v.

Random deep net assumption: In the ground truth, the edges between layers are chosen randomly subject to expected degree d being n^γ where $\gamma < 1/(l+1)$. In model $\mathcal{R}(l, \rho_l, \{G_i\})$, each edge carries a weight randomly chosen in $[-1, 1]$. In model $\mathcal{D}(l, \rho_l, \{G_i\})$, edge weights are chosen randomly in $\{-1, 1\}$. Also, expected density of last layer, $\rho_l(d/2)^l$ is assumed to be $O(1)$.

2.2 Main Results

Theroem 1 *When degree $d = n^\gamma$ for $0 \leq \gamma \leq 0.2$, density $\rho_l(d/2)^l = C$ for some large constant C, the network model $\mathcal{D}(l, \rho_l, \{G_i\})$ can be learnt using $O(\log n/\rho_l^2)$ samples and $O(n^2 l)$ time. The network model $\mathcal{R}(l, \rho_l, \{G_i\})$ can be learnt in polynomial time and using $O(n^3 l^2 \log n/\eta^2)$ samples, where η is the statistical distance between the true distribution and that generated by the learnt model.*

Authors give and analyse a new layerwise learning algorithm that exploits the fact that each layer is a denoising autoencoder. The algorithm boils down to finding correlations between nodes of a layer and putting them together this information globally to learn graph between layers. It leverages sparsity conditions of our generative model and uses well-known properties of random graphs, in particular, unique neighbors property. Algorithm 2.2 gives high level view of this algorithm

Algorithm 1 Highlevel Algorithm

Input: Sample y's generated by a deep network described in 2.1

Output: the network/encoder and decoder functions.

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

[1] In this review, we assume all layers have equal number of vertices for brevity as is done by authors in their extended abstract. The case where these are different is also analysed by authors and it's not too different from this case.

3 Each Layer is a Denoising Auto-encoder

Modern deep net research assumes that intermediate layers approximately preserve useful information and it should be able to go back and forth easily between layers. Following definition of denoising autoencoder encapsulates this.

Defintion 1 *An autoencoder consists of an decoding function $D(h) = s(Wh+b)$ and a encoding function $E(y) = s(W'y + b')$ where W, W' are linear transformations, b, b' are fixed and s is a nonlinear function that acts identically on each coordinate. The autoencoder is denoising if $E(D(h) + \eta) = h$ with high probability where h is drawn from the input distribution, η is a noise vector drawn form the noise distribution. The autoencoder is said to use weight tying if $W' = W^T$.*

Single layer random neural net is a denoising autoencoder if the output layer has low density $\rho d \ll n$.

Lemma 1 *If $\rho d < 0.05$, then the single-layer network above is a denoising autoencoder with high probability where the noise distribution is allowed to flip every bit independently with probability 0.01.*

This lemma relies on property of random graphs called *strong unique neighbour property.* For any node $u \in U$ and any subset $S \subset U$, let $UF(u, S)$ be the set of unique neighbours u with respect to S,

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

then following property holds for any fixed set S of size ρn with high probability over randomness of graph.

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

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

The set S has $(1-\epsilon)$-strong unique neighbour property if for every $u \in U$, u has $(1-\epsilon)$-unique neighbour property with respect to S.

If $\rho d \ll n$, for any fixed set of size S of size ρn, this property holds with high probability over the randomness of the graph.

Sketch of proof of lemma 1 : For a given network, definition of decoder is implicit: $y = \text{sgn}(Wh)$. Let the encoder be $E(y) = \text{sgn}(W^T y + 0.2d\bar{1})$. i.e, neural net run in reverse with threshold $0.2d$. Let S be support of h. This works because

(i) At least 0.9 fraction of the neighbours of node u in S are unique and about half of these are connected by a positive edges. Total weight for such edges is at least about $0.45d/2 = 0.225d$.

(ii) Each v not in S has at most $0.1d$ that are also neighbours of S. So, total weight of edges connected to these neighbours cannot be more than $0.1d$.

(iii) There is enough margin to allow for small noise. □

4 Learning Single Layer Network

Algorithm 2.2 outlined in Section 2.2 learns the network layer by layer starting from the bottom. Thus the key step is learning of a single layer network which is described in this section. The hidden and observed layer each have n nodes and generative model assumes that hidden layer is a random 0/1 assignment with ρn nonzeros.

4.1 Step 1 – Construct correlation graph

Say two nodes are *related* if they have a common neighbour in hidden layer to which they are attached by a positive edge. Aim of this step is to find all related nodes. This step is a new twist on Hebbian rule ("things that fire together wire together"). In the next step, this information about related nodes is used to recover all the positive edges.

```
Algorithm 2 Pairwise Graph
Input: N = $O(\log n/\rho)$ samples of $y = \mathrm{sgn}(Gh)$, where $h$ is unknown and chosen from uniform $\rho n$ sparse distribution
Output: Graph $\hat{G}$ on vertices $V$, $u, v$ are connected if u,v are related
for each $u, v$ in output layer do
  if there are atleast $2\rho N/3$ samples of $y$ satisying both $u$ and $v$ are fired then
    Connect $u$ and $v$ in $\hat{G}$
  end if
end for
```

Lemma 2 *If $\rho \ll 1/d^2$, 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 $(\rho d)^2$.*

Sketch of the proof : Consider that Let u and v are related. ie there is a vertex z such that uz and vz have positive weights. Let, $U = B(u) \cup B(v) \setminus \{z\}$. When $\mathrm{supp}(h) \cap U = \emptyset$ and $h_z = 1$, u and v both are surely fired. So,

$$P_h[u = 1, v = 1] \geq P[h_z = 1, \mathrm{supp}(h) \cap U = \emptyset] = P[\mathrm{supp}(h) \cap U = \emptyset] P[h_z = 1 |\, \mathrm{supp}(h) \cap U = \emptyset]$$

Due to simple concentration bounds, size of U cannot be too large than, say $3d$. So, first term, $P[\mathrm{supp}(h) \cap U = \emptyset] \geq 1 - 3\rho d \geq 0.9$. When $\mathrm{supp}(h) \cap U = \emptyset$, h is still ρm sparse. Hence $P[h_z = 1 |\, \mathrm{supp}(h) \cap U = \emptyset] \geq \rho$. This gives us $P_h[u = 1, v = 1] \geq 0.9\rho$.

Now consider that u and v are not related but both u and v fired. There should be nodes y and z that are 1 and are connected via positive edges to u and v respectively. Probability of this can be bound by $(\rho d)^2$ using union bound. Note that $(\rho d)^2 \ll \rho$. □

Now, we just need to estimate $P[u = 1, v = 1]$ up to a accuracy of, say $\rho/4$ which can be done using $O(\log n/\rho^2)$ samples by Chernoff bounds. So, algorithm 2 recovers all related nodes.

The assumption $\rho \ll 1/d^2$, might seem very strong. It can be made weaker by using higher order correlations. If 3-wise correlations are used, following algorithm works if $\rho \ll 1/d^{1.5}$. The proof that this algorithm works is very similar to that of pairwise case.

```
Algorithm 3 3-Wise Graph
Input: N = $O(\log n/\rho)$ samples of $y = \mathrm{sgn}(Gh)$, where $h$ is unknown and chosen from uniform $\rho n$ sparse distribution
Output: Hypergraph $\hat{G}$ on vertices $V$, $u, v, s$ is an edge if and only if they share a positive neighbour in G
for each $u, v, s$ in output layer do
  if there are atleast $2\rho N/3$ samples of $y$ satisying both $u, v$ and $s$ are fired then
    Connect $u, v$ and $s$ in $\hat{G}$
  end if
end for
```

4.2 Step 2 – Use RecoverGraph procedure to find all the positive edges

The problem is to recover graph given nodes at a distance at most 2 in the graph. This problem is a sub-case of Graph Square Root problem which is NP-hard. However, this is solvable in case of random graphs or random-like graphs.

Defintion 2 Graph Recovery problem: *There is a unknown random bipartite graph $G_1(U, V, E_1)$ between $|U| = |V| = n$ vertices. Every edge is chosen with probability d/n.*
Given : *Graph $\hat{G}(V, E)$ where $(v_1, v_2) \in E$ iff v_1 and v_2 share a common parent in G_1.*
Goal : *Find the bipartite graph G_1.*

Let $\Gamma(v)$ denote neighbours of v in $\hat{G}$. Proof of the Algorithm requires following properties :

1. For any $v_1, v_2 \in V$, $|(\Gamma(v_1) \cap \Gamma(v_2)) \setminus F(B(v_1) \cap B(v_2))| < d/20$. This property says that all except possibly $d/20$ neighbours of v_1 and v_2 in G_1 are caused by a common cause of v_1 and v_2

2. For any $u_1, u_2 \in U$, $|F(u_1) \cup F(u_2)| > 1.5d$. Since $F(u)$'s are chosen randomly, they should be fairly disjoint.

3. For any $u \in U, v \in V$ and $v \notin F(u)$, $|\Gamma(v) \cap F(u)| < d/20$. i.e, neighbour v unrelated to u cannot have too many correlation with neighbours of u

4. For any $u \in U$, at least 0.1 fraction of pairs $v_1, v_2 \in F(u)$ doesn't have a common neighbour other than u. This says that every cause introduces a significant number of correlations that is unique to itself.

5. For any $u \in U$, $0.8d \leq |F(u)| \leq 1.2d$.

All these properties hold whp over randomness of graph if $d^3/n \ll 1$

Algorithm 4 RecoverGraph

Input: $\hat{G}$ as given in definition 2
Output: 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
        $S' = \{v \in S : |\Gamma(v) \cap S| \geq 0.8d - 1\}$
        In $G_1$, create a vertex $u$ and 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 nodes are marked
```

Lemma 3 *When graph G_1 satisfies properties 1-5, Algorithm 4 successfully recovers the graph G_1 in expected time $O(n^2)$*

Proof: If (v_1, v_2) has more than one cause, say u_1 and u_2. Then S contains union of $F(u_1)$ and $F(u_2)$.By property 2, $|S|$ is at least $1.5d$ and condition in if statement is not satisfied.

Now suppose that (v_1, v_2) has a unique cause u, then we show that $S' = F(u)$. From property 1, $S = F(u) \cup T$ where $|T| < d/20$. So, $|S| < 11d/20$ and condition in if statement is satisfied.

For any vertex v in $F(u)$, $\Gamma(v)$ contains $F(u) \setminus \{v\}$. So, $|\Gamma(v) \cap S| \geq |\Gamma(v) \cap F(u)| \geq |F(u) \setminus \{v\}| \geq 0.8d - 1$. For any vertex v not in $F(u)$, $|\Gamma(v) \cap S| \leq |\Gamma(v) \cap F(u)| + |T| \leq d/10$ using property 3. So, $S' = F(u)$.

Algorithm will find all $u \in U$ because by property 4, there are enough number of correlations that are only caused by u. When such correlation is sampled in $\hat{G}$, u is recovered. So, all nodes in U are recovered in at most $10n$ iterations in expectation and each iteration takes $O(n)$ time. So, Algorithm runs in $O(n^2)$ time. □

If 3-wise correlation graph was constructed in previous step instead of pairwise graph, Algorithm 5 can be used to recover graph. This Algorithm gives better bounds. Proof of the algorithm runs similar to the previous case.

Algorithm 5 RecoverGraph3Wise

```
Input: Hypergraph $\hat{G}$ in which edges indicate correlation
Output: Underlying graph $G_1$
repeat
    Pick a random edge $(v_1, v_2, v_3) \in E$
    Let $S = \{v : (v, v_1, v_2), (v, v_1, v_3), (v, v_2, v_3) \in E\}$
    if $|S| < 1.3d$ then
        $S' = \{v \in S : |\Gamma(v) \cap S| \geq \binom{0.8d-1}{2}\}$
        In $G_1$, create a vertex $u$ and connect $u$ to every $v \in S'$
        Mark all the edges $(v_1, v_2, v_3)$ for $v_1, v_2, v_3 \in S'$
    end if
until all nodes are marked
```

4.3 Step 3 – Use the recovered positive edges to encode all samples

In step 2, we will have found edges which have positive edges. Following Algorithm allows us to recover assignment of hidden layer given assignment of output layer and positive edges.

Algorithm 6 PartialEncoder

```
Input: Positive edges $E^+$, sample $y = \mathrm{sgn}(Gh)$, threshold $\theta$
Output: the hidden variable $h$
  return  $h = \mathrm{sgn}((E^+)^T y - \theta\bar{1})$
```

Note that E^+ denotes 0/1 bipartite matrix not weight matrix.

Lemma 4 *If the support of vector h has the $11/12$-strong unique neighbour property in G, then Algorithm 6 return h given input E^+, $y = \mathrm{sgn}(Gh)$ and threshold $\theta = 0.3d$*

Proof: If $u \in S = \mathrm{supp}(h)$, at most $d/6$ neighbours can be shared with other vertices in S from strong unique neighbour property. Thus u has at least $0.3d$ neighbours which are 1.

If $u \notin S$, it can have intersection with at most $d'/4$ with $F(s)$, thus there cannot be more than $0.3d'$ of its neighbours that are 1. □

Steps $1, 2, 3$ work regardless of weights being discrete or continuous. This is because only the signs of weights are used in the proofs and algorithms.

4.4 Step 4 – Learn Graph

After step 3, the problem of unsupervised learning is turned into a supervised learning problem! So, weights of edges can be learnt to desirable accuracy if weights are real. if weights are in $\{-1, 1\}$, there is a simple algorithm to find all -1 edges.

Learning Discrete Weights

Now that pairs (h, y) have been given, the idea is to use all of them to determine all non-edges of the graph. Since all $+1$ edges are known, all -1 edges can be found.

Suppose in some sample, $y_v = 1$ for some v and exactly one neighbour of v in positive edge graph is in support of h. Let's call this neighbour u. There cannot be any -1 edge between v and $\text{supp}(h)$ as this would cancel out contribution of u. This motivates the following algorithm.

```
Algorithm 7 LearningGraph
Input: Positive edges $E^+$, samples of $(h, y)$, where h is from uniform $\rho n$-sparse distribution and $y = \text{sgn}(Gh)$
Output: $E^-$
$R \leftarrow (U \times V) \setminus E^+$
for each of the samples $(h, y)$ and each $v$ do
    Let S be the support of $h$
    if $y_v = 1$ and $S \cap B^+(v) = u$ for some $u$ then
        for $s \in S$ do
            remove $(s, v)$ from R
        end for
    end if
end for
```

Lemma 5 *Suppose we have $N = O(\log n/(\rho d^2))$ samples of pairs (h, y) with uniform ρm-sparse h and $y = \text{sgn}(Gh)$. Then with high probability over choice of the samples, Algorithm 6 outputs the correct set E^-.*

Sketch of the proof: All we really need to show is that probability that a non-edge is not identified is very low. It is not too tough to show that probability that a non edge (z, u) is identified by one sample (h,y) is at least $\rho d^2/3$. Thus the probability that it is not identified after $O(\log n/(\rho d^2))$ is $< 1/n^c$. By union bound, all non-edges are found whp. □

Let's summarise the results in this section into following theorem. 3-wise correlations are used to get bounds in this theorem.

Theroem 2 *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 $O(logn/\rho^2)$ samples to recover the ground truth with high probability over the randomness of the graph and the samples.*

Learning Continuous Weights

When the weights on the edges are continuous in $[-1, 1]$, learning the decoder becomes harder. In fact, an example can be constructed in which we cannot hope to learn the weights exactly without exponential number of samples. Thus improper learning is necessary when weights are real. An algorithm exists which achieves a slightly weaker guarantee: the decoder learned by this algorithm is correct with probability $1 - \eta$.

```
Algorithm 8 LearningDecoder
Input: $N = O(n^3 l^2 \log n/\eta^2)$ samples $(h^1, y^1), (h^2, y^2), \ldots, (h^N, y^N)$ where $y = \mathrm{sgn}(Gh)$
Output: A graph $G'$ such that $P_{h\sim\mathcal{D}_t}[\mathrm{sgn}(G'h) \neq \mathrm{sgn}(Gh)] \leq \epsilon$
  Solve the linear program
      $\forall j, G'h^j \leq 0$ if $y^j = 1$ and $G'h^j > 0$ if $y^j = 0$
  return a feasible solution $G'$, the docoder is $y = D(h) = \mathrm{sgn}(G'h)$
```

Lemma 6 *Given $N = O(n^3 l^2 \log n/\eta^2)$ samples of (h, y) where h is chosen from distribution $\mathcal{D}_t$ and $y = \mathrm{sgn}(Gh)$, with high probability over the choice of samples, Algorithm 8 outputs a matrix G' that satisfies $y^i = \mathrm{sgn}(G'h^i)$ for all samples (y^i, h^i). Furthermore,*

$$P_{h\sim\mathcal{D}_t}[\mathrm{sgn}(G'h) \neq \mathrm{sgn}(Gh)] < \eta/l$$

Proof: LP in Algorithm 8 is feasible because G is a feasible solution. Every feasible solution is consistent with all the samples.

Note that hypothesis class for coordinate y_v is simply all the halfplanes $\mathrm{sgn}((Gh)_v)$. By VC-dimension theory, since the VC-dimension of a half-plane is n+1, any hypothesis that is consistent with all N samples has generalization error $O(\sqrt{n \log N/N})$. More precisely,

$$P_{h\sim\mathcal{D}_t}[\mathrm{sgn}(G'^v h) \neq \mathrm{sgn}(G^v h)] \leq \sqrt{2n \log N/N} + O(\log n/2N) \leq \eta/nl$$

for each output node v. By union bound on all nodes, $P_{h\sim\mathcal{D}_t}[\mathrm{sgn}(G'h) \neq \mathrm{sgn}(Gh)] < \eta/l$

5 Correlations in a Multilayer network

We now consider multilayer networks. To extend the analysis of the previous section to multilayer networks, all that really needed is to prove that Algorithms 2 and 7 works even when distribution is not uniformly ρn sparse but is of distribution that is generated at an intermediate layer of neural network. Once this is confirmed, rest of the steps work as before without any need of modifications as only randomness of G_t is assumed in these steps.

Lemma 7 *For any u, v in the layer of $h^{(1)}$, if they have a common positive neighbour in the layer of $h^{(2)}$*

$$P[u = 1, v = 1] \geq \rho_2/2$$

Otherwise,

$$P[u = 1, v = 1] \leq \rho_2/4$$

Here, $\rho_i = \rho_{i+1} d_i/2$ is the expected density of layer i.

Sketch of the proof: Firstly, for any i $P[h^{(i)}(u) = 1]$ can be shown to be between $3\rho_i/4$ and $5\rho_i/4$.

Suppose that u, v in the layer of $h^{(1)}$ has a common neighbour z in layer $h^{(2)}$ with positive edges to the both and none of the neighbours with negative edge are 1 in the layer $h^{(2)}$. This condition ensures $u = 1, v = 1$. The probability of this can be shown to occur with the probability more than $\rho_2/2$ using the above bound and the fact that $B(\{u, v\})$ cannot be too large than $2d$.

Now, suppose that u and v are not related, It turns out that $P[u = 1, v = 1]$ can be bound by $\rho_2/5$ by using union bounds. □

This lemma is analogous to lemma 2 and it shows that Algorithm 2 works.

Lemma 8 *Suppose $\rho_y d \ll 1$ and h is chosen from distribution $\mathcal{D}_2$, using $O(\log n/(\rho^2 d))$ samples, with high probability over choice of the samples, Algorithm 7 returns E^-*

Proof of this is very similar to that of lemma 5.

These two lemmas allows us to learn layer by layer with little modification in steps of learning single layer network.

6 Learning the Lowermost (real-valued) Layer

Note that in our model, the lowest (observed) layer is real-valued and doesn't have threshold gates. So, our earlier learning algorithms cannot be applied as is. However, same paradigm - identifying correlations and using GraphRecover - can be used.

First step is to show that for a random weighted graph G, the linear decoder $D(h) = Wh$ and the encoder $E(y) = sgn(W^T y + b)$ form a denoising autoencoder with real-valued outputs.

Lemma 9 *If G is a random weighted graph, the encoder $E(y) = \mathrm{sgn}(W^T y - 0.4d\bar{1})$ and linear decoder $D(h) = Wh$ form a denoising autoencoder,for noise vectors 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.

Lemma 10 *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 neighbour, then $|E_h[y_u y_v y_s]| \leq \rho_1/3$*
2. *If u,v and s have a unique common neighbour, then $|E_h[y_u y_v y_s]| \geq 2\rho_1/3$*

Proof of this lemma is similar to that of lemma 2.

7 Two layers are more expressive than one

Authors finally show that a two layer network cannot be represented by a one layer network.

Lemma 11 *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 accommodated in a single-layer network even using arbitrary weights.

8 Conclusions

Authors give provably fast algorithms to learn random and sparse deep neural nets. They leverage properties of random graphs and sparsity to prove these.

Many aspects of deep nets are mysterious to theory: reversibility, why use denoising autoencoders, why this highly non-convex problem is solvable, etc. Authors 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.

Authors also note that random neural nets do seem useful in so-called reservoir computing.

References

[1] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. CoRR, abs/1310.6343, 2013.

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

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

[4] Daniel Hsu and Sham M. Kakade. Learning mixtures of spherical gaus- sians: moment methods and spectral decompositions. In Proceedings of the 4th conference on Innovations in Theoretical Computer Science, pages 1120, 2013.

[5] Sanjeev Arora, Rong Ge, and Ankur Moitra. 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, pages 110, 2012.

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

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

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

[9] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In ICML, pages 10961103, 2008.

[10] Yoshua Bengio. Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2(1):1127, 2009. Also published as a book. Now Publishers, 2009.