Algorithms Reading Group Notes: Provable Bounds for Learning Deep Representations

Joshua R. Wang

November 1, 2016

1 Model and Results

Continuing from last week, we again examine provable algorithms for learning neural networks. This time, we will consider the following problem: we draw a random ground truth network and choose some random inputs for it. Running these inputs through the model, we generate outputs. Given just these outputs, can we efficiently compute a network which is statistically (in ℓ_1 norm) indistinguishable from the ground truth network? It turns out that we can, by leveraging the structure in the random graphs between layers of the network.

The work considers several cases for the ground truth network distribution. In one simple case, there are ℓ hidden layers $h^{(\ell)}, \ldots, h^{(1)}$ and one observed layer y. Each layer has exactly n nodes. Edges between each layer are chosen randomly subject to the expected degree d being n^γ, with $\gamma < 1/(\ell+1)$. Edges have weights random in $\{\pm 1\}$. Hidden nodes have a boolean value, while the output layer is real-valued. We impose the restriction that the expected density of the last layer $\rho_\ell d^\ell / 2^\ell$ is $O(1)$. We denote this distribution over ground truth networks as $\mathcal{D}$. [1]

The input distribution is uniform over $\rho_\ell n$-sparse boolean vectors.

Theorem 1.1. *When degree $d = n^\gamma$ for $0 < \gamma \leq 0.2$ and density $\rho_\ell (d/2)^\ell = C$ for some large constant C, the network model $\mathcal{D}$ can be learned using $O(\log n / p_\ell^2)$ samples and $O(n^2 \ell)$ time.*

If the weights are instead drawn from $[-1, +1]$ then we would require $O(n^3 \ell^2 \log n / \eta^2)$ samples and polynomial time, where η is the total variation distance between the distribution generated by the output network and the true distribution for all but the last layer.

2 Denoising Autoencoders

It is often convenient if a deep net (or at least several layers of it) approximately preserve information. This naturally leads to the concept of an *autoencoder*, which aims to capture one way this "reversbility" property can occur.

Definition. *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 said to use weight tying if $W' = W^T$.

Possible functions for s include logistic functions and soft max. Today, s will be a simple threshold function.

For more robust reversibility, we can force the autoencoder to reconstruct the input from a corrupted version of the output. This results in a *denoising autoencoder*:

[1] This distribution can be generalized, but the resulting description and analysis become more complex. We cover this simple case today, which is the focus of the short version of the paper.

Definition. *An autoencoder is denoising if $E(D(h) + \xi) = h$ with high probability where h is drawn from the distribution of the hidden layer and ξ is a noise vector drawn from the noise distribution.*

Note that normally we would want to minimize the reconstruction error $||y - D(E(y+\xi))||$, but for today our y will be exactly $D(h)$.

Our ground truth distribution $\mathcal{D}$ makes each layer of the network a denoising autoencoder with high probability. We now prove this fact:

Lemma 2.1 (Layers are Denoising Autoencoders)**.** *If the input layer is a random ρn sparse vector and noise flips every output bit independently with probability 0.1, then a single layer of a network in $\mathcal{D}$ is a denoising autoencoder with high probability. It uses weight tying.*

Before proving our Layers are Denoising Autoencoders Lemma, we first cover some notation for and properties of random bipartite graphs:

Definition. *Let $G = (U, V, E, W)$ be a bipartite graph with vertex sets U and V, edges E, and weights W.*
For any node $u \in U$, let $F(u)$ be its neighbors in V.
For any subset $S \subseteq U$, let $UF(u, S)$ be the unique neighbors of u with respect to S, i.e.,

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

A node $u \in U$ has the $(1-\epsilon)$-unique neighbor property with respect to a set $S \subseteq U$ if:

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

A set $S \subseteq U$ has the $(1-\epsilon)$-strong unique neighbor property if every node in U has the $(1-\epsilon)$-unique neighbor property with respect to S.

Fix a set $S \subseteq U$ of size ρn. We will use the result that with high probability over the randomness of the graph, S has the $(1-\epsilon)$-strong unique neighbor property without proof. The intuition is that it holds due to Hoeffding's Inequality and a union bound. We can now prove our Layers are Denoising Autoencoders Lemma:

Proof. The decoder is $D(h) = \text{sgn}(Wh)$, and the encoder is $E(y) = \text{sgn}(W^T y + b')$ where $b' = -0.2d\overrightarrow{1}$.

To see why these two functions work, consider the set of nodes which are 1 in the input layer. A majority of their neighbors in the output layer are unique neighbors. Any unique neighbors with positive edges will be 1, because there are no negative edges to cancel. Hence, by looking at the set of nodes which are 1 in the output, you can infer the correct assignment to the input by doing a threshold of $0.2d$ at each node. □

3 Learning a Single Layer

The plan for a single layer $h^{(i)} \to h^{(i-1)}$ proceeds as follows:

1. Construct a *correlation graph* using samples of $h^{(i-1)}$.
2. Use RecoverGraph to learn the positive edges E_i^+ from the correlation graph.
3. Use PartialEncoder to encode all $h^{(i-1)}$ to $h^{(i)}$.
4. Use LearnGraph to learn the negative edges E_i^-.

Step 1: Construct the Correlation Graph

Our plan is to first identify *related* nodes in the observed layer: two nodes which have a common neighbor in the hidden layer to which they are attached via a +1 edge. To do so, we begin by constructing a correlation graph over output nodes. In particular, we connect pairs of nodes $u, v \in V$ that are both one in at least a $\rho/3$ fraction of our samples.

Proposition 3.1 (Correlation Graph)**.** *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. Suppose that u, v are a related pair and z a vertex with +1 edges to both. Let S be the set of other neighbors of u, v. S cannot be much larger than $2d$. By our choice of parameters, $\rho d \ll 1$, so the event that S is all zeros conditioned on z is 1 is at least 0.9. But the probability that z is 1 is ρ, so the probablity of both u and v being 1 is at least 0.9ρ.

Now suppose that u, v are an unrelated pair. For both u, v to be 1 there must be nodes y and z, both of which are 1, and which are connected to u and v respectively via +1 edges. By a union bound, this event has probability at most $(\rho d)^2$. □

Hence if $(\rho d)^2 < 0.1\rho$, then $O(\log n/\rho)$ samples should suffice to recover all pairs w.h.p. This assumption can be weakened if we instead use 3-wise correlations; call a triple of observed nodes related if they are connected to a common node via +1 edges. This allows us to prove an analagous claim about recovering all related triples correctly, but we end up with a 3-uniform hypergraph instead. RecoverGraph can be modified to work correctly with triples.

Step 2: Recover Positive Edges

Notation note: in this subsection, we pretend only positive edges exist, and d refers to the positive degree only.

We want to recover the positive edges of the bipartite graph (U, V, E, W) from the correlation graph. To do so, we will rely on the following properties which are all satisfied by random graphs w.h.p. when $d^3/n \ll 1$:

1. For any two observed nodes $v_1, v_2 \in V$, consider their common neighbors in the correlation graph. At most $d/20$ of these do not share a common positive parent.
2. For any two hidden nodes $u_1, u_2 \in U$, they have at least $1.5d$ distinct positive children together (not much overlap).
3. For any hidden $u \in U$ and observed $v \in V$ which is not a positive child of u, there at most $d/20$ neighbors of v in the correlation graph which are positive children of u.
4. For any hidden $u \in U$, at least a 0.1 fraction of pairs v_1, v_2 from its positive children do not have a common neighbor other than u.

The algorithm is as follows. We repeatedly do the following until all edges of our correlation graph have been marked:

- Pick a random edge (v_1, v_2) in the correlation graph.
- Compute S, the set of common neighbors of v_1 and v_2 in the correlation graph.
- Abort if $|S| \geq 1.3d$.
- Compute S', the set of nodes in S who are neighbors with at least $0.8d - 1$ other nodes in S. Note: S' should be a clique in the correlation graph.
- Create a node u which is connected to every node in S' by a positive edge.

- Mark all the edges between nodes in S'.

Lemma 3.2. *If the graph of positive edges satisfies the properties above, then our algorithm recovers it in $O(n^2)$ expected time.*

Proof. By Property 2, we abort if (v_1, v_2) have more than one common positive parent, since then $|S|$ will be at least $1.5d$.

By Property 1, if (v_1, v_2) have a unique common positive parent, then we do not toss out any children of this parent when pruning from S to S'.

By Property 3, if (v_1, v_2) have a unique common positive parent, then we toss out every node which is not a child of this parent when pruning from S to S'.

By Property 4, it takes 10 iterations in expectation to identify a new node $u \in U$. Since each iteration takes at most $O(n)$ time, this algorithm runs in $O(n^2)$ time. □

Step 3: Recover Hidden Variables

We now want to recover the hidden variables using the observed variables and positive edges. Let M be the adjacency matrix of positive edges. We should compute h as follows:

Lemma 3.3 (PartialEncoder)**.** *If the support of h satisfies the 11/12-strong unique neighbor property, and $y = sgn(Wh)$, then choosing $h = sgn(M^T y - \theta \overrightarrow{1})$ with the threshold $\theta = 0.3d$ outputs h.*

Proof. For any hidden node z with $h_z = 1$, it has at least $0.4d$ unique neighbors that are connected with $+1$ edges. This implies all these neighbors are 1, so $[M^T y]_z \geq 0.4d$.

On the other hand, if $h_z = 0$, z has few common neighbors with the 1's in h (at most $0.2d$, say).

Taken together, a threshold of $0.3d$ suffices. □

Step 4: Recover Negative Edges

Now all that remains is to compute the negative edges. Consider the following situtation in a sample (h, y). Suppose for some observed node v, $y_v = 1$ but there is exactly one hidden node u with the property that $h_u = 1$ and there is a $+1$ edge between them. In other words, in this sample v is receiving exactly $+1$ positive weight total, so it cannot receive any negative weight. Hence any other hidden nodes which are 1 in the sample must not have a -1 edge to v.

Our algorithm is to simply go through all samples (h, y) and rule out edges as above. We claim that this suffices to recover the negative edges:

Lemma 3.4 (LearnGraph)**.** *Given $O(\log n/(\rho^2 d))$ samples of pairs (h, y), with high probability (over the random graph and the samples) we recover the correct set of edges E^- after ruling out E^+ and nonedges as above.*

Proof. Consider a non-edge (x, u). We would like these three events to happen: (i) $h_x = 1$, (ii) u has exactly one positive edge to a hidden nodes in h which are 1, and (iii) u has no negative edges to hidden nodes in h which are 1. These events are almost independent and occur with probabilities roughly ρ, ρd, and 1, respectively. □

4 Learning Multiple Layers

So, as it turns out, we can't simply apply this single layer algorithm repeatedly. The problem is that we have assumed that each hidden layer is a random ρn-sparse vector, when it in fact there are correlations between nodes due to previous layers. Note that by the unique neighbor property, we can expect that roughly $\rho_\ell(d/2)$ fraction of $h^{(\ell-1)}$ to be 1, and hence roughly $\rho_\ell(d/2)^{\ell-i}$ of $h^{(i)}$.

Lemma 4.1 (Multilayer Correlations)**.** *Consider a network from $\mathcal{D}$. With high probability (over 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. We say a node s at the topmost layer is an ancestor of u if there is a path of length $\ell - 1$ from s to u. u has roughly $2^{\ell-1}\rho_1/\rho_\ell$ ancestors. With good probability, at most one ancestor of u has value 1. A node is a positive ancestor of u if, when only that node is 1 at the topmost layer, 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 on, then either one of their common positive ancestors is on, or both of them have a positive ancestor that is on. The probability of the latter is $O(\rho_1^2)$ which by assumption is much smaller than ρ_2.

If u, v have a common positive parent z, then the positive ancestors of z are all common positive ancestors of u, v, so there are at least ρ_2/ρ_ℓ. The probability of exactly one being on is at least $\rho_2/2$.

If they do not have a common parent, then the number of common positive ancestors depends on the number of common grandparents. With high probability over the graph structure (not shown here), the number is small and hence the probability that they are both 1 is small. □

This Multilayer Correlations Lemma is what we actually need to guarantee that our threshold of $\rho_2/3$ works to give us the correlation graph. Once we have that, the other steps proceed as normal. We can then repeat this argument by peeling off the bottom layer.

5 Learning the Final Layer

Recall that the final layer is real-valued, and hence does not directly work with these algorithms for boolean-valued layers. However, the same plan of computing the correlation graph and using it to recover the positive edges, hidden variables, and negative edges still works.

Lemma 5.1. *The encoder $E(y) = sgn(W^T y - (0.4d)\overrightarrow{1})$ and linear decoder $D(h) = Wh$ form a denoising autoencoder for noise vectors γ which have independente components, each having variance at most $O(d/\log^2 n)$*

Theorem 5.2 (Final Layer Correlation Graph)**.** *When $\rho_1 d = O(1)$, $d = \Omega(\log^2 n)$, with high probability over the choice of 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, s have no common neighbor, then $|\mathbb{E}_h[y_u y_v y_s]| \leq \rho_1/3$*
2. *If u, v, s have a unique common neighbor then $|\mathbb{E}_h[y_u y_v y_s]| \geq 2\rho_1/3$*

Note that there is a more complex statement of the Final Layer Correlation Graph Theorem which corresponds to the more nuanced analysis of Section 4.

6 Experimental Insights

The authors tested their algorithm on *synthetic data*, and concluded the following:

- The most expensive step was finding the negative edges, but it was possible to stop early and reasonably over-estimate the negative edges.
- Trying to learn a network with two hidden layers ($\ell = 2$) with $n = 5000$, $d = 16$, and $\rho_\ell n = 4$ took $500,000$ samples and less than one hour to learn the first layer and positive edges of the second layer; learning negative edges of the second layer requires more samples according to the analysis, but the algorithm only made 10 errors with $5,000,000$ samples.

References

[ABGM14] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *ICML*, pages 584–592, 2014.