COMS 4774 Spring 2021 Project Report

Harris Sean (sh3715)

Department of Computer Science, Columbia University

April 21, 2021

Abstract

This is my project report for COMS 4774 Spring 2021.

1. **Title:** Provable Bounds for Learning Some Deep Representations
2. **Authors:** Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma
3. **Bibliographic information:** CoRR abs/1310.6343, 2013
4. **Link:** `http://arxiv.org/abs/1310.6343`

1 Paper review

1.1 Brief summary

The goal of this paper is to provide an algorithm with provable time and sample complexity for learning a special family of neural nets: random and sparse generative nets.

1.2 Problem statement

Generic neural nets are trained by feeding an input through the net's layers, and then backpropogating a loss vector in order to calculate a weight gradient which will improve the loss function. For multilayer nets with large inputs, the dimension of the loss function may be enormous, and learning ideal weights through supervised learning is "badly NP-hard". A different strategy for learning of the net is inspired by the paradigm of generative neural nets, where a neural net is run 'in reverse' to generate data similar to the input distribution.

In this strategy, studied by the authors, learning is done in an unsupervised, layer-wise manner. First the bottom-most layer is shaped according to the unlabelled data, which is then used as data to shape the next layer, and so on, until the net is a good generative model for the original distribution.

The unsupervised strategy avoids the NP-hardness of gradient descent, but problems remain. There is no known criteria for judging whether the net preserves information and filters noise between layers - acting as a so called denoising auto-encoder. Furthermore, at the time of writing there were limited provable bounds of learning layers in this unsupervised fashion.

1.3 Main results

The ground-truth neural net has ℓ hidden layers $h^{(\ell)}, h^{(\ell-1)}, ..., h^{(1)}$ and an observed layer y at the bottom. Each layer i has n_i binary vertices, which connect to layer $i+1$ along edges E_i. Importantly, the graph connecting $h^{(i)}$ and $h^{(i+1)}$ is at most degree $d = n^{\gamma}$, where γ is a small constant $< 1/(\ell+1)$. In order to generate random samples, the top layer $h^{(\ell)}$ is initialized with a 0/1 assignment where the set of nodes set to 1 is picked uniformly among all sets of size $n\rho_\ell$.

Two different nets are considered: $\mathcal{R}(\ell, \rho_\ell, G_i)$ which has edge weights $G_{i,j,k} \in [-1, 1]$, and $\mathcal{D}(\ell, \rho_\ell, G_i)$ which has discrete weights $G_{i,j,k} \in \{-1, 1\}$.[1] $\mathcal{D}(\ell, \rho_\ell, G_i)$ can be learned using $O(\log n/\rho_\ell^2)$ samples, running in time $O(\sum_{i=1}^{\ell} n_i(d_i^3+n_{i-1}))$. $\mathcal{R}(\ell, \rho_\ell, G_i)$ can be learned using $O(n_\ell^2 n_{\ell-1}\ell^2 \log n/\eta^2)$ samples, running in polynomial time.

Algorithm 1 High Level Algorithm

Input: samples $y_i ... y_N$ from ground-truth net

Output parameters of ground-truth net

```
0: for i = 1 TO $\ell$ do
1:     Call FindCorrelations to create correlation graph using samples of $h_{(i)}$
2:     Call RecoverGraph to recover positive edges of $G_i$
3:     Call PartialEncoder to recover $h_{(i+1)}$ from $h_{(i)}$ for all samples of $h_{(i)}$
4:     Call LearnGraph to recover negative edges of $G_i$
4: end for =0
```

1.4 Key ideas and insights

1.4.1 Denoising Auto-Encoders

A denoising auto-encoder is a pair of functions that preserve information and filter noise when data is passed through them. One function might encode data to a more compressed form, and the other decodes the compressed data into the original format. Formally, there is an encoder $E(y) = s(W^\top y + b^\top)$and a decoder $D(h) = s(Wy + b)$, where s is some nonlinear function like sign(x), W is a weight matrix, and b is a bias vector.

The authors prove each layer of the family of neural nets they analyze act as denoising auto-encoders with high probability. The proof relies on the unique-neighbor property of a sparse random graph - the idea being that since connections are sparse, connections are unique, so sub-communities of nodes in the graph are isolated from each other. If communities are distinct, then the output nodes in y fire exactly when the parent node in h fires, so $E(D(h)) = y$. This might be surprising, considering the net's weights are random. However, the weighted bipartite graph G_i between two layers is essentially just a random matrix transformation of some input vector, with a piece-wise nonlinear function applied at the end. So the authors' findings mirror well known properties of random matrix projection, such as they preserve distance between points with high probability.

1.4.2 Correlation based unsupervised learning

Contrary to generic supervised training of neural nets with gradient descent, where the net's weights are adjusted according to labelled data, the authors provide a novel unsupervised algorithm to learn a net's weights. Given samples of a generative neural net, the algorithm first infers positive output edge-weights based on the pairwise or 3-wise correlations in the samples. This is done by

[1] G_i is used interchangeably for both the bipartite graph and the bipartite weight matrix of the graph at layer i.

constructing an adjacency graph $\hat{G}$, where $\hat{G}[i,j] = 1$ if the ith and jth node in the samples fire together enough times - on the order of ρN, where N is the number of samples and ρ is the layer density.

Algorithm 2 FindCorrelations

```
Require: $N = O(\log n/\rho^2)$ samples of $y = (Gh)$, where $h$ is unknown and chosen from uniform
  $\rho m$-sparse distribution
Ensure: Hypergraph $\hat{G}$ on vertices $V$. $\{u, v, s\}$ is an edge if and only if they share a positive
  neighbor in $G$
  for each $u, v, s$ in the observed layer of $y$ do
    if there are at least $2\rho N/3$ samples of $y$ satisfying all $u, v$ and $s$ are fired then
      add $\{u, v, s\}$ as an hyperedge for $\hat{G}$
    end if
  end for=0
```

With the adjacency graph, the positive output edge-weights are found by first identifying a community, then pruning any members who are not “well connected” enough in the community, with a threshold on the order of d', the density of the forward edges. This has to be done because some unrelated nodes may happen to correlate with some members of the community, but likely not all or most.

Algorithm 3 RecoverGraph

```
Require: Hypergraph $H$ in Definition
Ensure: Graph $G_1$ in Defintion
  repeat
    Pick a random hyperedge $(v_1, v_2, v_3)$ 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
      Let $S' = \{v \in S : v$ is correlated with at least $\binom{0.8d'-1}{2}$ pairs in $S\}$
      In $G_1$, create a vertex $u$ and connect $u$ to every $v \in S'$.
      Mark all hyperedges $(v_1, v_2, v_3)$ for $v_1, v_2, v_3 \in S'$
    end if
  until all hyperedges are marked =0
```

Recall the ith layer $h^{(i)}$ is found by computing $h^{(i)} = sgn(G_i h_{i+1})$. Having acquired the positive edges of G_i, we can determine w.h.p. the vector $h^{(i+1)}$ from $h^{(i)}$. Even though the negative edges are unknown, if enough child nodes of some parent in $h^{(i+1)}$ are activated, w.h.p. that parent is activated. This also relies on the unique-neighbor property.

Algorithm 4 PartialEncoder

```
Require: positive edges $E^+$, $y = (Gh)$, threshold $\theta$
Ensure: the hidden variable $h$
  Let $M$ be the indicator matrix of $E^+$ ($M_{i,j} = 1$ iff $(i, j) \in E^+$)
  return $h = (M^T y - \theta\vec{1})$ =0
```

With the positive edges of G_i, $h^{(i+1)}$, and $h^{(i)}$, the negative edges of G_i can be inferred. The central idea here is that if some child node is activated, and it is connected to some activated parent node, given the unique-neighbor property, the child is unlikely to be connected to any other activated node in $h^{(i+1)}$. Very few parent nodes are activated, and each parent has very few children, so the probability of some child having two activated parents is low. With enough samples of $h^{(i+1)}$ and $h^{(i)}$, w.h.p. all of these non-edges can be found. Since the positive edges are known, the remaining unknown edges must all be negative.

Algorithm 5 Learning Graph

```
Require: positive edges $E^+$, samples of $(h, y)$, where $h$ is from uniform $\rho m$-sparse distribution,
    and $y = (Gh)$
Ensure: $E^-$
 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 $s \in S$ do
 6:       remove $(s, v)$ from $R$.
 7:     end for
 8:   end if
 9: end for
10: return $R$ =0
```

This entire algorithm is then repeated, using the inferred parent layer $h^{(i+1)}$ as the new child layer to find correlations within. This is repeated for all layers, until the random neural net is learned.

1.5 Strengths and weaknesses

This algorithm learns a generative neural net in polynomial time, with a reasonable amount of samples. This approach avoids NP-hard supervised gradient descent, using a relatively simple and novel algorithm that can be implemented using basic linear algebraic operations. However, the provable bounds guaranteeing this algorithm depend on the neural net being entirely random, and the properties of a random bipartite graph, notably the unique-neighbor property. This algorithm could not learn, for instance, the parameters of a net trained to generate human faces, as there is no guarantee of 1. sparse layers, 2. limits on the number of forward edges, or 3. random uniform edge weights, all of which are necessary for the unmodified algorithm to halt, let alone correctly learn the net.

However, there are some advantages to random neural nets. As mentioned, w.h.p. a layer of the ground truth net is a denoising auto-encoder. The authors also mentioned practical application of random neural nets in reservoir computing.

2 Empirical Experiments

2.1 Demonstration of the algorithm

This section details an implementation of the author's algorithm in Python using Numpy. A single layer of $\mathcal{D}(\ell, \rho_\ell, G_i)$ was generated, then samples $y = sgn(Gh)$ were generated, and the authors' algorithm was run to recover the neural net from the samples.

Each entry of the tables below contains two measures of how accurate the recovered net is compared to the ground-truth net. The first number is the percent of columns of the weight matrix G

exactly recovered, which is also the forward edges of each parent node. The second number measures how the recovered net behaves statistically: a new batch of samples $\hat{X_1}, ..., \hat{X_N}$ are generated from the recovered net, which are compared to the ground-truth samples $X_1, ..., X_N$. The metric chosen to compare the distributions is the Frobenius norm of the difference between the ground-truth covariance matrix Σ and the reconstructed covariance matrix $\hat{\Sigma}$. The provided error is this value normalized by the magnitude of Σ:

$$accuracy = \frac{|\Sigma|_F - |\Sigma - \hat{\Sigma}|_F}{|\Sigma|_F}$$

All nets shown had input density $\rho = 3/n$.

25 Node Net					
	100 samples	200 samples	500 samples	1000 samples	5000 samples
degree 4	**4%**, .28	**12%**, .37	**16%**, .42	**4%**, .32	**8%**, .36
degree 5	**4%**, .26	**4%**, .37	**0%**, .35	**4%**, .42	**12%**, .41
degree 7	**0%**, .29	**4%**, .31	**4%**, .31	**4%**, .39	**0%**, .45
degree 9	**0%**, .29	**0%**, .36	**0%**, .36	**0%**, .3	**0%**, .34
degree 11	**0%**, .22	**0%**, .23	**0%**, .24	**0%**, .28	**0%**, .25

50 Node Net					
	100 samples	200 samples	500 samples	1000 samples	5000 samples
degree 4	**2%**, .24	**20%**, .33	**32%**, .39	**56%**, .46	**50%**, .52
degree 5	**4%**, .29	**18%**, .33	**26%**, .42	**32%**, .48	**48%**, .52
degree 7	**0%**, .27	**8%**, .36	**18%**, .4	**24%**, .45	**32%**, .45
degree 9	**0%**, .27	**6%**, .32	**2%**, .37	**4%**, .31	**8%**, .35
degree 11	**0%**, .28	**0%**, .28	**2%**, .33	**10%**, .35	**2%**, .32

125 Node Net					
	100 samples	200 samples	500 samples	1000 samples	5000 samples
degree 4	**8%**, .12	**62%**, .31	**86%**, .50	**93%**, .64	**100%**, .83
degree 5	**11%**, .10	**64%**, .34	**87%**, .52	**96%**, .65	**100%**, .84
degree 7	**1%**, .02	**25%**, .25	**41%**, .27	**40%**, .33	**34%**, .33
degree 9	**7%**, .10	**11%**, .16	**11%**, .13	**10%**, .12	**11%**, .13
degree 11	**4%**, 0	**9%**, .15	**18%**, .16	**9%**, .08	**12%**, .16

The exact weight matrix G was recovered when $d = 4, 5$ in a 125 node net, given more than 1000 samples.

2.2 Sparse neural connections

The parameters were chosen in order to remain on the order of the bounds provided by the authors' proofs, while also remaining practical. The main bounds on the parameters are below.

1. $\rho > 1/n$
2. $\rho \ll d^{-3/2}$
3. $\binom{.8d-1}{2}$ should exist

4. For all parent nodes u, $|F(u)| \in [.8d, 1.2d]$ where $F(u)$ is the set of forward nodes of u

Bound (1) comes from the fact that $h^{(i+1)}$ must have atleast one activated node. Bounds 2-4 are needed to recover the edges of G. Since the density ρ cannot be lower than one node out of n, this limits how large the degree d of the graph can be in practice. Furthermore, to find threewise correlations, a parent node needs atleast three children, but since we prune members of communities with less than $\binom{.8d-1}{2}$ neighbors, d needs to be atleast 5 for $\binom{.8d-1}{2}$ to be greater than 1. However, on average half of edges have negative weights, meaning they won't contribute to correlations, and are "invisible" to the positive graph edge recovery. In order to get around this, low degree nets were forced to have atleast 4 positive edges per node, so the algorithm could halt.

The limits imposed on the parameters of the net, usually related to the randomness or sparsity of the weight matrix, is a limitation of the direct practical application of this algorithm. However, the paper sets out to provide theoretical provable bounds on learning random nets, and this empirical test both demonstrates its success under correct conditions, and can give an idea of what kind of empirical neural networks could be learned with this algorithm or future work in the same vein.

The sparsity of the recovered net is also an advantage, as the algorithm will learn a sparse net that generates the sample distribution provided, even if that sample distribution comes from a dense net with more layers. It is well known that generic neural nets often contain far more parameters than needed for their task, and can be pruned to a smaller subgraph of the original net with similar performance. The authors' algorithm in a sense comes pre-pruned, and if a sparse net is capable of generating the sample distribution, the algorithm will return it.

2.3 Each layer is a denoising auto-encoder

The authors demonstrate the layers of ground-truth neural net w.h.p. are denoising auto-encoders, such that $E(D(h)+n) = h$ for some noise vector n. Denoising auto-encoders approximately preserve information and filter noise. This is a property expected of neural nets in general, and the authors' proof serves as some theoretical justification for this assumption.

Below is an experiment to measure how a layer from the paper's ground-truth random neural effects an input vector, using the MNIST handwritten digits dataset of 28×28 pixel images, which can be flattened into an input vector $h \in \mathbb{R}^{784}$. The random weighted bipartite graphs $G_1, G_2, G_3 \in \mathbb{R}^{784\times784}$ were generated such that every entry took a value from the uniform distribution $[-1, 1]$ with probability $\frac{degree}{n}$, where the degree was chosen to be 15. A one layer pass

$$G_1^\top(G_1h) - \theta\vec{1}$$

was calculated , shown in Figure 2, along with a three layer pass

$$G_1^\top(G_2^\top(G_3^\top(G_3(G_2(G_1h))) - \theta\vec{1}) - \theta\vec{1}) - \theta\vec{1}$$

shown in Figure 3. The threshold constant θ was set to $\frac{degree}{3}$. The MNIST images differ in important ways from the sparse, random vectors used in the paper's proof. First, they are not random, and second, there is no guarantee $\rho d \ll 1$. Even still, the image vectors remain identifiable after one pass through the encoder and decoder function.

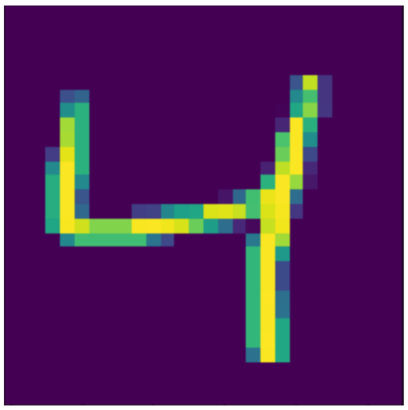

Figure 1: Original image

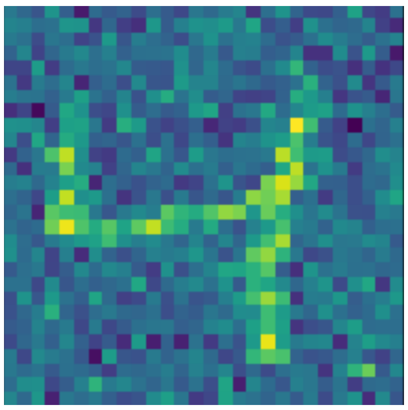

Figure 2: one layer

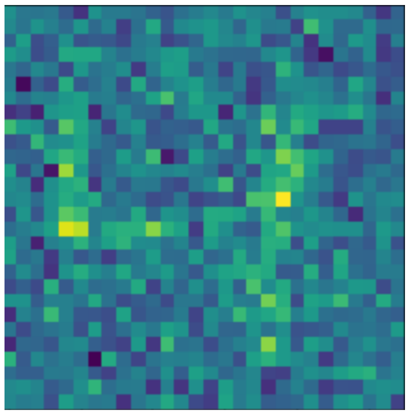

Figure 3: Three layers