Self Similarity, Contraction Mappings, and Distributed Deep Learning of Neural Networks*

Priyanka Kargupta† Kamalika Das‡ Hillol Kargupta§

Abstract

Distributed computing and its applications in many domains such as machine learning and data analysis typically benefits from problem decomposability. If a problem can be decomposed into sub-problems and the sub-problems can be solved efficiently followed by the aggregation of the results from the sub-problems then distributed computing often scales up. This paper explores the notion of self-similarity and contraction mappings for constructing distributed representations in the context of deep neural networks. It offers a mathematical framework for constructing local similarity preserving representations using randomized contraction mappings and their Banach fixed points. It also notes that the gradient descent error minimization learning algorithms (e.g. back-propagation algorithm) for deep neural networks can be reduced to the problem of computing covariance matrix in a distributed environment. The paper exploits locally self-similar representations for designing communication efficient distributed inner product computation which in turn is used for estimating covariance matrix from data observed at different nodes. The results presented in this paper can also be used for constructing a decomposed problem representation even in machine learning problems from centralized data. The paper also presents experimental results in order to support the theoretical analysis of the algorithm presented here.

1 Introduction

Decomposing a problem into a set of sub-problems, solving these sub-problems in parallel, and combining their results for putting together the final solution is the essence of many scalable problem solving methodologies. Distributed computing and its applications in the field of machine learning is no exception. Self-similarity and contraction mappings offer a fundamental approach toward creating a decomposable representation. Self-similarity, in plain language, represents structures that exhibit a similar kind of shape or similar properties when a part of the original structure is zoomed in. Consider Figure 1, which shows the self-similar structures in a leaf. By definition, a self-similar set can be divided among a collection of subsets where each of these subsets is similar to each other and the original set. A self-similar object may be represented by the smaller parts of the object. For example, the fern leaf shown in Figure 1 may be represented using smaller self-similar components of the leaf. In fact this observation is the foundation behind fractal encoding and compression algorithms [2]. This paper explores the application of self-similar structures in distributed decomposed computation of statistical properties relevant to machine learning algorithms such as deep auto-encoder neural networks.

Figure 1: Self-similar structures in a leaf.

This paper claims that machine learning problems can be efficiently solved in a decomposed and distributed manner by creating self-similar representations of the problem. The basic idea is that in order to learn a function or a classifier involving an object like the fern leaf shown in Figure 1, we may not need the entire leaf. Since this leaf has self-similar structures, we may be able to learn the function or the classifier just by using a small part of the leaf structure that is similar to the whole structure–thereby reducing the computational and communication cost of the machine learning problem. This research develops a mathematical framework to do so using randomized contraction mappings. Contraction mappings [1] over a metric space intuitively "contracts" the members of the domain. For example, if we apply a contraction mapping to couple of locations on a map then the distance between the transformed

*Partially supported by DOT Contract DTRT5716C10001
†Mt. Hebron High School, Ellicott City, MD 21043
‡USRA, NASA Ames, Mountain View, CA 94035
§Agnik, 8840 Stanford Blvd, Ste 3500, Columbia, MD 21045

locations will be less than the original distance between those two locations. The new distance between the two points will be reduced by a scaling factor less than 1. An interesting property of contraction mappings is that they are guaranteed to have a fixed point. In other words, if we apply the contraction mapping repeatedly on any member of the domain then they will all converge to a unique point known as the fixed point of the contraction mapping. This fixed point also stays invariant when the contraction mapping is applied on that fixed point. This property is also known as the Banach fixed point theorem of contraction mappings. The paper points out that the fixed point of a contraction mapping can also be used to accurately compute statistical properties such as covariance in a distributed environment.

This paper specifically considers the problem of distributed deep learning of neural networks, showing that learning in neural networks can be reduced to the covariance or inner product computation problem. It develops a special class contraction mapping — randomized contraction mappings — and offers an algorithm to learn the covariance structure of the data using the fixed points of the local data under the randomized contraction mappings. The methodology offers a highly communication efficient algorithm to learn deep multi-layer auto-encoder/decoder neural network.

Section 2 summarizes the main contribution of this paper. Section 3 offers the background material needed for this paper. Section 4 discusses various useful results regarding self-similarity and contraction mappings. Section 5 develops the notion of randomized contraction mappings. Section 6 explores randomized contraction mappings in the context of auto-encoder neural networks. Section 7 discusses distributed learning of auto-encoder neural networks using randomized contraction mappings. Section 8 presents the experimental results. Finally, Section 9 concludes this paper.

2 Contribution

This paper blends mathematical results from many different areas and develops its own results. It draws motivation from the topology literature in the area of self-similarity and fractals. It also exploits results from randomized embeddings literature such as large random vectors are on average orthogonal to each other. The paper combines many of these results along with our current understanding of distributed computing for machine learning and data mining. The main contributions of this work are as follows:

1. Develop a mathematical framework for constructing randomized contraction mappings.
2. Exploit the fixed points of randomized contraction mappings in order to estimate statistical properties such as covariance.
3. Combine auto-encoder neural network with randomized contraction mappings in order to create the so called ANRC (Auto-Encoder Neural network with Randomized Contraction) networks.
4. Prove that ANRC networks are contraction mappings on average.
5. Prove that the incorporation of randomized contraction mappings in an auto-encoder neural network does not change the structure of the weight matrix on average. In other words, the network topology and weights stay invariant on average even after the randomized contraction mapping is applied to the input data.
6. Develop a distributed algorithm for learning the weights in an auto-encoder network.

The following section presents a review of the related literature.

3 Background

Constructing a decomposed representation of a problem has been close to the heart of artificial intelligence and machine learning research from the very early days of the field. Simon's [10] discussion on the architecture of complexity pointed out the need for quasi-decomposable representations for efficient problem solving. Holland [7] offered a similar perspective in natural and artificial systems. The field of neural networks has also been deeply connected with distributed and parallel processing right from the beginning. For example, one of the early work in multi-layer neural network by Rumelhart [9] underscored the parallel and distributed processing of information in natural and artificial neural networks. There exists a relatively large body of literature on distributed processing of neural networks. There also exists a large body of literature on distributed machine learning and data mining [8]. The following part of this section presents a brief overview of the relatively recent literature of the work on multi-layer neural networks.

Dean et al. [4] used a distributed approach involving DistBelief, a framework utilizing model parallelism within and across machines with the details of parallelism, synchronization, and communication managed by the framework. Within the framework, two algorithms were developed for large-scale distributed training: (i) Downpour SGD, asynchronous stochastic gradient descent procedure which leverages adaptive learning rates and supports a large number of model replicas and (ii) Sandblaster L-BFGS, a distributed implementation

of the L-BFGS algorithm that uses both data and model parallelism.

Black et al. [5] details the basics of distributed training of neural networks and various algorithms used for training, some of which include parameter averaging, asynchronous stochastic gradient descent with soft synchronization protocols, and decentralized stochastic gradient descent.

Dong et al. [6] proposes a DDRL model, a hierarchical structure set up on distributed resources designed to abstract the layer-wise image feature information.The model extracts random patches on the multiple Map nodes, pre-processes using PCA, uses K-means for learning the dictionary, employs spatial pooling on the extracted features, and produces feature maps that are inputted to the second layer where k-means is used again. The last layer uses the dictionary to extract image features to use as input to train the SVM for classification.

Zhang et al. [12] keeps track of the staleness associated with each gradient computation and adjusts the learning rate on a per-gradient basis by simply dividing the learning rate by the staleness value using the proposed ASGD algorithm. The algorithm automatically tunes the learning rate based on gradient staleness using an N-softsync protocol and achieves model accuracy comparable with SSGD while providing a near-linear speedup in runtime.

Strom [11] introduces a new method for scaling up distributed stochastic gradient descent training of deep neural networks that controls the rate of the weight-update per individual weight which reduces the amount of communication by three orders of magnitude while training a typical DNN for acoustic modelling.

Chen et al. [3] demonstrates that synchronous optimization with backup workers can avoid asynchronous noise while mitigating for the worst stragglers as asynchronous stochastic optimization emphasizes speed by using potentially stale information for computation which can result in poorer trained models.

The work presented here is very different from the existing body of literature listed above. The current work explores a mathematically well-grounded methodology to bound the communication among different nodes for computing statistical properties. The following section discusses the foundation material for self-similar sets and contraction mapping.

4 Self Similarity and Contraction Mapping

Most practical supervised and unsupervised machine learning problems can be represented as function learning problems. Learning functions can be efficient and scalable when the function can be decomposed into a set of sub-functions so that each sub-function can be computed efficiently in parallel. The results are then aggregated to produce the final result. Note that since functions can be viewed as sets in the function space, we can use a set theoretic framework whenever appropriate without losing any generality.

Self-similar sets offer some interesting properties that may allow efficient computation of functions over these sets. Let us first define self-similarity in mathematical terms and then explore some of these properties.

Definition 4.1. (Infinitely Separable Self-Similarity) *A set F is self-similar to each subset f_i with respect to a characteristics $\Psi(.)$ if $\Psi(F) \sim \Psi(f_i)$ and F is infinitely separable, i.e.*

$$\begin{aligned} F &= \{f_1^1, f_2^1, \cdots f_i^1 \cdots f_n^1\}, \\ f_i^1 &= \{f_1^2, f_2^2, \cdots f_i^2 \cdots f_n^2\}, \\ &\cdots \\ f_j^k &= \{f_1^{k+1}, f_2^{k+1}, \cdots f_i^{k+1} \cdots f_n^{k+1}\} \textit{where } k+1 \to \infty \end{aligned}$$

Consider an indicator function $I_\psi(F)$ that returns a value of 1 if $\Psi(F)$ is true and 0 otherwise for the set F. Since the characteristic property stays invariant across all the subsets f_i^k, one may evaluate the function $I_\psi(F)$ by evaluating $I_\psi(f_i^k)$. If necessary, further statistical significance can be gained by evaluating $I_\psi(f_i^k)$ for different values of i and k. In other words, for any self similar set F, a task that involves evaluating the indicator function $I_\psi(F)$ can be accomplished by using any subset of the self-similar subsets f_i^k.

Note that if $|F|$ and $|f_i^k|$ denote the cardinalities of the sets F and f_i^k respectively then $F = |f_1^1| + |f_2^1| + \cdots |f_i^1| \cdots + |f_n^1|$ and $|f_i^1| = |f_1^2| + |f_2^2| + \cdots + |f_i^2| + \cdots |f_n^2|$. Therefore, $|F| \geq |f_i^k|$. In other words, evaluation of the indicator function can potentially be performed using a smaller number of data points compared to the original set F. This is main idea behind this paper—If the data set has self-similarity then the self-similar subsets can be used to compute statistical quantities. Although self-similarity is widely prevalent in many data sets, one could philosophically argue even a stronger claim—unless there is local self-similarity and pattern in the data, problem solving (particularly machine learning) is not computationally tractable [10]. We will revisit this observation later in this paper in order to develop communication efficient distributed machine learning algorithms.

Next, we consider contraction mappings and relate those with self-similarity. Contraction mappings provide useful properties for creating similarity preserving decomposed representations.

Definition 4.2. (Contraction Mapping) *If* (Θ, d)

and (Θ', d') are two metric spaces, a contractive mapping is a function $\theta : \Theta \to \Theta'$ such that $d'(\theta(x), \theta(y)) \leq cd(x, y)$ for all $x, y \in \Theta$ and the contractivity factor $c < 1$.

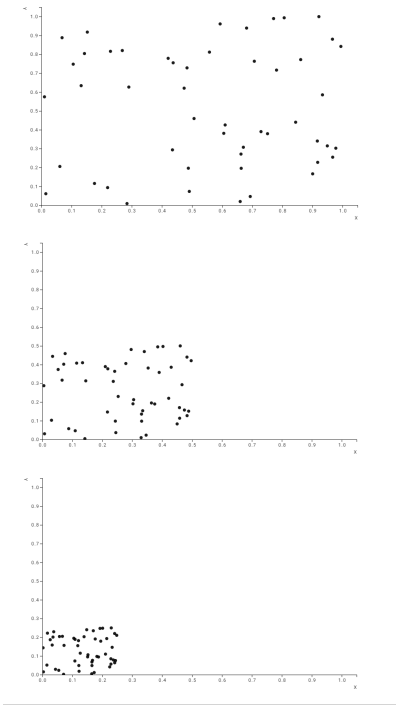

Figure 2: (Top) Original data X set in a 2D space. (Middle) The transformed data set $\theta(X)$. (Bottom) Twice transformed data set $\theta\theta(X)$). The graphs show the gradually contracting data toward the fixed point.

Consider a two dimensional Euclidean space and a contraction mapping defined by $\theta(x, y) = (\frac{x}{2}, \frac{y}{2})$. Figure 2 (top) shows a set of randomly chosen data set where $x, y \in [0, 1]$. Figures 2 (middle and bottom) shows the data after the contraction mapping is applied once and twice.

By Banach's fixed point theorem, a contractive mapping is guaranteed to a unique fixed point $\overline{u}$ such that $\Theta(\overline{u}) = \overline{u}$. If we are interested in finding a contractive mapping that maintains the invariance of a given target point v then we can pose the problem as follows:

DEFINITION 4.3. *Given an element $v \in F$, find a contraction mapping $\Theta : F(X) \to F(X)$ with fixed point $\bar{u}$ such that $d(v, \bar{u})$ is as small as possible.*

The following theorem (known as Collage Theorem [2] has found many applications in different domains and it is also useful in the context of this current paper.

THEOREM 4.1. *Given an element $v \in F$ and a contracion mapping $\Theta : F(X) \to F(X)$ with fixed point $\bar{u}$ and a contractivity factor c,*

$$d(v, \overline{u}) \leq \frac{1}{1-c} d(v, \Theta v)$$

This allows us to pose the problem as a search for a contractive mapping Θ so that the Collage error $d(v, \Theta v)$ is minimized.

LEMMA 4.1. *Given $v_1, v_2 \in F$ and a contracion mapping $\Theta : F(X) \to F(X)$ with fixed point $\bar{u}$ and a contractivity factor c,*

$$d(v_1, v_2) \leq \frac{1}{1-c}(d(v_1, \Theta v_1) + d(v_2, \Theta v_2))$$

Proof: By triangle inequality

$$d(v_1, v_2) \leq d(v_1, \overline{u}) + d(v_2, \overline{u})$$

Using Theorem 4.1, we can write

$$\begin{aligned} d(v_1, v_2) &\leq \frac{1}{1-c} d(v_1, \Theta v_1) + \frac{1}{1-c} d(v_2, \Theta v_2) \\ &= \frac{1}{1-c}(d(v_1, \Theta v_1) + d(v_2, \Theta v_2)) \end{aligned}$$

■

For a given set of points F, one can generalize the problem of finding a set of invariant domain members to the problem of finding a set of contractive mappings $\overline{\Theta} = \Theta_1, \Theta_2, \cdots \Theta_N$ where each Θ_i is defined over F such that for a unique set $\overline{A} = \{A_1, A_2, \cdots A_N\}$ and $\overline{A} \subset F$,

$$A_i = \Theta(A_i)$$

Using the set notation, we can write

$$\overline{\Theta}(\overline{A}) = \overline{A}. \tag{4.1}$$

In other words, set $\overline{A}$ is self-similar. Equation 4.1 can be used to generate self similar subsets of F using contraction mappings.

5 Randomized Contraction Mappings

This section explores a special class of contraction mappings that are generated using randomized algorithms. It specifically considers linear transformations defined over an inner product space where entries of the transformation matrix are random variables. The paper makes use of these randomized contraction mappings later in order to develop distributed algorithms for training deep neural networks.

THEOREM 5.1. (RANDOMIZED CONTRACTION MAPPING) *Let V be an $m \times k$ dimensional matrix where the $(i,j)-th$ cell of V, $V_{i,j} \in \{-\sqrt{c/k}, \sqrt{c/k}\}$ with uniform probability. On average, V is a contraction mapping if $0 \leq c < 1$.*

Proof: Note that the expected value,

$$E[VV^T] = cI$$

Where I is the identity matrix.

If X is an $m \times n$ dimensional matrix where each row is drawn from an inner product space, let $X' = V^T X$. Therefore,

$$\begin{aligned} E[(X')^T X'] &= E[X^T V V^T X] \\ &= X^T (cI) X \\ &= c(X^T X) \end{aligned}$$

Therefore, V is a contraction mapping.

■

Let us now explore what we can do with these contraction mappings, specifically the randomized contraction mappings. The rest of this section argues that fixed points of randomized contraction mappings can be effectively used to estimate the covariance matrix. For mean zero random variables, we can write: $Cov(x,y) = \frac{1}{m}\sum_{k=1}^{m} x_k y_k$.

Now, consider a contraction mapping θ defined over the 2-dimensional space created by the two random variables x and y. Let $(\overline{x}, \overline{y})$ be the values of the random variables x and y respectively, corresponding to the fixed point of the contraction mapping θ. Therefore, as we apply the contractive mapping repeatedly on the domain members,

$$Cov(x,y) \rightarrow \frac{1}{m}\sum_{k=1}^{m} \overline{x}.\overline{y} = \overline{x}.\overline{y}$$

6 Learning an Autoencoder Neural Network

Consider a linear autoencoder as shown in Figure 3. It embeds the input feature vector $\mathbf{X_1}$ to $\mathbf{X_2}$ using a linear mapping such that $\mathbf{X_2} = \mathbf{W_2 X_1}$. Usually, this encoding step embeds the input feature vector to a lower dimensional space. The next step is to decode that to the output space $\mathbf{Y} = \mathbf{W_1 X_2}$. Therefore $\mathbf{Y = W_1 W_2 X_1}$.

THEOREM 6.1. *If $X' = V^T X$ and $\mathbf{Y = W_1 W_2 X_1}$ represents an autoencoder network and V is an $m \times k$ dimensional matrix where the $(i,j)-th$ cell of V, $V_{i,j} \in \{-\sqrt{c/n}, \sqrt{c/n}\}$ with uniform probability then a linear autoencoder network is a contraction mapping when the input data is transformed using the matrix V.*

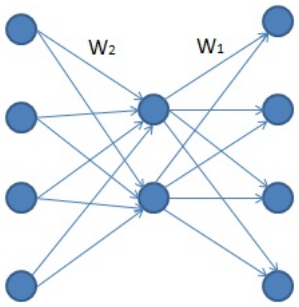

Figure 3: Autoencoder network with weight vector W_1 and W_2 in the outer and inner layer respectively.

Proof:

$$\begin{aligned} \mathbf{Y^T Y} &= \mathbf{X'}_1^T \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{W}_1 \mathbf{W}_2 \mathbf{X'}_1 \\ &= \mathbf{X'}_1^T (\mathbf{W}_1 \mathbf{W}_2)^T (\mathbf{W}_1 \mathbf{W}_2) \mathbf{X'}_1 \\ &= \mathbf{X'}_1^T \mathbf{X'}_1 \\ &= (V^T \mathbf{X}_1)^T (V^T \mathbf{X}_1) \end{aligned}$$

Therefore, we can write:

$$E[\mathbf{Y^T Y}] = E[\mathbf{X}_1^T V V^T \mathbf{X}_1] = \mathbf{X}_1^T cI \mathbf{X}_1 = c\mathbf{X}_1^T \mathbf{X}_1$$

■

The same result is also true for non-linear auto-encoders where the output values of a neuron are passed through a nonlinear function such as a logistic or a RELU function.

THEOREM 6.2. *If $X' = V^T X$ and V is an $m \times k$ dimensional Randomized Contraction Mapping matrix where the $(i,j)-th$ cell of V, $V_{i,j} \in \{-\sqrt{c/n}, \sqrt{c/n}\}$ with uniform probability then a non-linear autoencoder network is a contraction mapping when the input data is transformed using the matrix V.*

Proof:

Consider a non-linear auto-encoder where $X_2 = W_2 X_1$ and the non-linear transformation is introduced by $X_3 = G(X_2)$. Similarly, for the outer layer, $X_4 = W_1 X_3$ and $Y = G(X_4)$. We can write, $X_3 = G(W_2 X_1)$, $X_4 = W_1 G(W_2 X_1)$, $Y = G(W_1 G(W_2 X_1))$. Since it is an auto-encoder, $Y = X_1$. Therefore, we can write:

$$E[\mathbf{Y^T Y}] = E[\mathbf{X}_1^T V V^T \mathbf{X}_1] = c\mathbf{X}_1^T \mathbf{X}_1$$

■

So far, we have proved that both linear and non-linear auto-encoder neural networks can be viewed as

a contraction mapping when the input data is transformed using a randomized contraction mappings. Going forward, we shall use the term Auto-Encoder Neural network with Randomized Contraction (ANRC) for implying this type of network architecture. In the following discussion, we prove that the weight matrix of an ANRC network is on average identical to a regular auto-encoder neural network with a small modification in the learning equation.

Most multi-layer neural networks reported in the literature learn the weights associated with the neurons by gradient descent on the error surface. For example, the Back-propagation learning law [9] works by minimizing the error function,

$$E(W) = \frac{1}{2N}\sum_{k=1}^{N}||T^k - O(X^k, W)||^2$$

Weight update equation for the single i-th node in the output layer can be written as:

$$\begin{aligned} w_i^{(t+1)} &= w_i^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(t^{(k)} - \sum_j w_{i,j}^{(t)} x_j^{(k)})x_i^{(k)} \\ &= w_i^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(t^{(k)}x_i^{(k)} - \sum_j w_{i,j}^{(t)} x_j^{(k)}x_i^{(k)} \end{aligned}$$

Weight update equation for the complete outer layer can be written using matrix notation as follows:

$$\begin{aligned} \mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(\mathbf{T}^{(k)}\mathbf{X}^{(k)} - \\ &\qquad \mathbf{W}^{(t)}(\mathbf{X}^{(k)})^T\mathbf{X}^{(k)}) \end{aligned} \tag{6.2}$$

We can now prove the following theorem.

LEMMA 6.1. *If V is a randomized contraction mapping matrix so that $V_{i,j} \in \{-\sqrt{c/n}, \sqrt{c/n}\}$ with uniform probability, the weight matrix of a ANRC network learned based on Equation 6.2 remains invariant when the input data set $\mathbf{X}$ is replaced by $\mathbf{V}^T\mathbf{X}$, corresponding target output is $\mathbf{V}^T\mathbf{T}^{(k)}$, and the learning rate is $\frac{\eta}{c}$.*

Proof:

$$\begin{aligned} \mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} + \frac{\eta}{cN}\sum_{k=1}^{N}(\mathbf{T}^{(k)}\mathbf{V}\mathbf{V}^T\mathbf{X}^{(k)} - \\ &\qquad \mathbf{W}^{(t)}(\mathbf{X}^{(k)})^T\mathbf{V}\mathbf{V}^T\mathbf{X}^{(k)}) \end{aligned}$$

Recall that $E[\mathbf{V}^T\mathbf{V}] = cI$. Substituting that in the above equation, we get,

$$\begin{aligned} E[\mathbf{W}^{(t+1)}] &= \mathbf{W}_i^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(T^{(k)}\mathbf{X}^{(k)} - \\ &\qquad \mathbf{W}_i^{(t)}(\mathbf{X}^{(k)})^T\mathbf{X}^{(k)}) \end{aligned}$$

The above equation is identical to Equation 6.2.

■

The above result can be easily proved for the learning equation in the hidden layers. The proof is very similar to the one presented here for the output layer. Now that we have proved that the weight matrix of the ANRC network is identical to that of the regular auto-encoder network on average, we can conclude that the introduction of the randomized contraction mapping for transforming the data matrix does not change the outcome of the learned model on average. In the following section, we argue that the randomized contraction mapping however, provides a major benefit in creating a distributed decomposed representation of the underlying learning problem and thereby results in scalable communication efficient machine learning of auto-encoder networks.

7 Distributed Autoencoder Network Learning

This section considers the problem of learning an autoencoder network in a distributed environment. Consider a distributed environment with p data sources. Let $\mathbf{X_i}$ be the row vector of features observed from the i-th data source. $\mathbf{X_1}, \mathbf{X_2} \cdots \mathbf{X_p}$ be the features vectors observed at data sources $1, 2, \cdots p$ respectively. Let n_i be the number of features observed at source i and $\sum_{i=1}^{p} = n$. The weight update equation for multiple node scenario can be written as follows:

$$\begin{aligned} \mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(T^{(k)}[\mathbf{X}_1^{(k)} \cdots \mathbf{X}_p^{(k)}] - \\ &\qquad [\mathbf{W_1^{(t)}} \cdots \mathbf{W_p^{(t)}}][\mathbf{X}_1^{(k)} \cdots \mathbf{X}_p^{(k)}]^T[\mathbf{X}_1^{(k)} \cdots \mathbf{X}_p^{(k)}]) \\ &= \mathbf{W}^{(t)} + \frac{\eta}{N}\sum_{k=1}^{N}(T^{(k)}[\mathbf{X}_1^{(k)} \cdots \mathbf{X}_p^{(k)}]) - \\ &\qquad \eta[\mathbf{W_1^{(t)}} \cdots \mathbf{W_p^{(t)}}]Cov(\mathbf{X}) \end{aligned}$$

Where $Cov(\mathbf{X}^{(k)})$ is the sample covariance matrix computed over the training data set after a mean-zero translation. Therefore, the weight update equation can be reduced to the following items:

- The $\sum_{k=1}^{N} T^{(k)}[\mathbf{X}_1^{(k)} \cdots \mathbf{X}_p^{(k)}]$ term and
- $Cov(\mathbf{X})$

The first item is a linear term and can be additively decomposed among the p nodes and then aggregated in $O(p)$ communication complexity. The second term involves computation of the covariance matrix and efficient computation of that requires further discussion.

7.1 Distributed Covariance Computation Note that the computation of $Cov(\mathbf{X})$ requires computing the inner product of the input feature vector observed at different nodes. The naive approach computes the inner product between two n dimensional vectors stored at different nodes in $O(n)$ communication. Bring elements of the data vector to a single node in $O(n)$ and then compute the inner product in $O(n)$ time. Over k training examples, estimation of a single term in the covariance matrix requires $O(kn)$ communication.

Table 1: Naive randomized algorithm for inner product computation

1. Let $\delta_i \in \{-1, 1\}$ with equal probability.
2. Repeat the following steps for $j = 1, 2, \cdots m$.
   Compute $\mathbf{A}' = A_1\delta_1, A_2\delta_2, \cdots A_n\delta_n$ and $\mathbf{B}' = B_1\delta_1, B_2\delta_2, \cdots B_n\delta_n$
   Compute $S_{A,j} = \sum_i A'_i$ and $S_{B,j} = \sum_i B'_i$
   Send the scalar data item S_A to the node l_B and compute $I_j = S_{A,j}S_{B,j}$.
3. Compute $P = \frac{1}{k}\sum_{j=1}^{k} I_j$. P is an estimator of the inner product between $\mathbf{A}$ and $\mathbf{B}$.

7.2 Naive Randomized Algorithm for Covariance Estimation The inner product between two data vectors $\mathbf{A} = A_1, A_2, \cdots A_n$ and $\mathbf{B} = B_1, B_2, \cdots B_n$ observed at two different nodes l_A and l_B can be computed with much less communication cost using different techniques. Table 1 shows a simple randomized algorithm to compute the inner product between two vectors observed at two different nodes. The algorithm simply flips the sign of the data items in each vector identically with uniform probability. This can be accomplished by using the same seed for the pseudo-random number generator at each site. The randomly perturbed data items are then added up and the resulting scalar is sent to the other site or to the central site that is computing the inner product. The two scalars are multiplied and averaged over m trials. The final averaged result is an estimator for the inner product of the two vectors $\mathbf{A}$ and $\mathbf{B}$. This randomized algorithm estimates the inner product in $O(m)$ communication.

7.3 Contraction Mapping for Covariance Estimation The problem with the naive algorithm presented in the previous section is that it does not exploit advantage of any self-similarity in the data set. The contraction mapping-based framework discussed in this paper can be effectively used to further reduce the communication further by exploiting the inherent self-similarities in the data set used for learning the auto-encoder network. Rest of this section discusses the algorithm.

Recall from Theorem 4.1 that $d(v, \overline{u})$ is upper bounded. Also, from Theorems 6.1, 6.2, and Lemma 6.1 we observe that an ANRC network minimizes the collage error for the data tuples in the training data set. Therefore, the contraction mapping of an ANRC network must satisfy the inequality in Theorem 4.1. Table 2 shows the algorithm for distributed covariance estimation.

Table 2: Covariance estimation.

1. Initialize set $S = \emptyset$.
1. Generate a set of randomized contraction mappings $V_1, V_2, \cdots V_k$.
2. Select the contraction mapping V_i that minimizes the collage error over the given data set X.
3. Compute the fixed point $\overline{u}_i$ of V_i and set $S = S \cup \overline{u}_i$.
4. Compute an estimate of the covariance matrix using the members of the set S. Repeat steps 1, 2, and 3 until the covariance matrix converges.

7.4 Distributed Auto-Encoder Learning Table 3 shows the pseudo code for the proposed algorithm. The distributed ANRC algorithm essentially exploits the self-similarity in the data set using randomized contractive mappings. It selects the optimal mapping that minimizes the collage error over subsets of the training data sets. If the auto-encoder is trained using an already available training data set in the batch mode then data subsets can be selected using various schemes. The simplest scheme would be a random selection of data points with uniform probability. However, that approach fails to recognize the local self-similarity properties often exhibited in many applications such as image processing. Therefore, dividing the data points based on proximity makes sense. For example, one may select data points for a subset by dividing the image in a grid structure and choosing the points that fall within a single cell of the grid for one subset. Each subset would correspond to a cell in the grid. Fixed points for each subset is computed and shared with the remote nodes for computing the cross-terms in the network weight matrix.

If the data set is not available in the batch mode and the learning problem is essentially an online incremental one then one may have to incrementally build the estimates of the covariance matrix needed for computing the cross-terms in the weight matrix. As new data points come in, the new updates of the fixed points for

the randomized contraction mappings become available and once the fixed points converge, those can be propagated to the remote nodes.

Table 3: Distributed learning in an ANRC network.

1.	Construct an ANRC network.
2.	Select a representative set of randomized contraction mappings and update the local weights as usual.
3.	Send the fixed points of the contraction mappings to the remote sites and update the coss-terms in the weight.
4.	Repeat steps 2 and 3 until the network converges.

8 Experimental Results

This section presents experimental results in order to back up the analytical claims of this paper. Figure 4 (Top) shows the original data set in a 10 dimensional space. It shows the density plot of the data. Figure 4 (Bottom) shows the density plot of the transformed data set using a randomized contraction mapping. Figure 5 (Top) shows the correlation matrix of the original data set. Figure 5 (Bottom) shows the correlation matrix computed using the fixed points of the transformed data set. As the figure shows, the estimation of the covariance matrix (we show the normalized correlation matrix since it has normalized values) obtained using the proposed approach appears to be good.

The following part of this section explores a larger data set widely used for image classification benchmarking using deep neural networks. Tiny Imagenet has 200 classes of animals, humans, buildings, and other objects. Each class has 500 training images, 50 validation images, and 50 test images. Both class labels and bounding boxes are provided as annotations for the training and validation sets. In order to test the distributed algorithms, we introduced copies of these images after making different linear transformations (e.g. rotation, translation, scaling) and distributed the different transformed copies of the same image among different nodes. The goal is to create a test environment with multiple nodes observing different but related images. For example, if I represents the original image and $I_1, I_2, \cdots I_n$ represent the different copies of I under various linear transformations then these transformed images are distributed among different compute nodes. Amazon Web Service (AWS) EC2 instances (g2.2x.large) with GPU modules are used for this experiment. We used a hub and spoke distributed computing architecture for the experiments. Since a pure auto-encoder network is an unsupervised learning network, Figure 6 shows the variation of accuracy with respect to the amount of data communicated through fixed points.

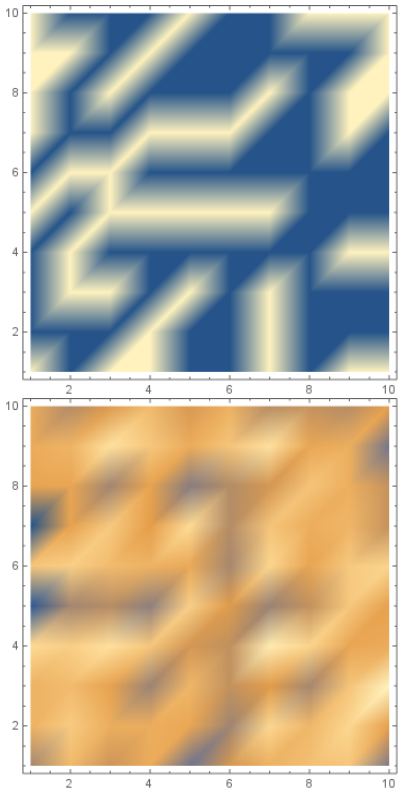

Figure 4: (Top) Original data set in a 10 dimensional space. (Bottom) The transformed data set using a randomized contraction mapping.

9 Conclusions

This paper explored the fundamental problem creating distributed and decomposed representation of functions in the context of machine learning—specifically for learning auto-encoder neural networks. It noted that decomposing a problem among sub-problems and distributed problem solving requires identifying the patterns in the data and exploiting those to construct an appropriate representation. Self-similarity is a common phenomenon observed in many data set. Particularly in image classification involving natural scenes self-similarity is often abundant. The paper borrowed ideas from topology theory, dynamical systems, and randomized algorithms in order to develop a distributed algorithm for learning auto-encoders.

The paper developed the notion of randomized contraction mapping and showed that an auto-encoder neural network along with input layer transforming the input data using randomized contraction mapping min-

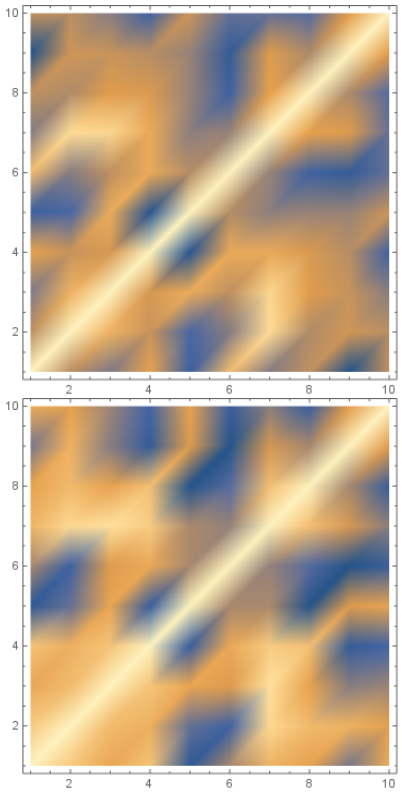

Figure 5: (Top) Correlation matrix of the data presented in Figure 4. (Bottom) Correlation matrix of the transformed data using the randomized contraction mapping.

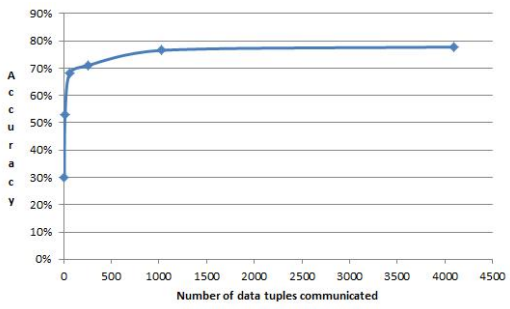

Figure 6: (Accuracy vs. number of fixed points communicated per node. A classifier is learned using the ANRC network and the corresponding accuracy on the test data set is reported. Results show that highly accurate classifiers can be learned by communicating a small number of fixed points generated using the grid-based subset selection approach described in this paper.

imizes the collage error over the training data points. The paper also proved that this does not however change the topology and the values of weight matrix on average. This is desirable since this property makes sure that the introduction of randomized contraction mapping for developing a distributed decomposed version of the learning problem does not change the functional behavior of the original problem. The developed distributed algorithm makes use of the fixed points of the randomized contraction mappings in order to efficiently learn the auto-encoder network.

This work opens up many new future directions. More detailed analytical work bounding the variance of the expectation-based results presented in this paper and extending the results beyond auto-encoder networks are some examples.

References

[1] Banach, S. (1922). Sur les oprations dans les ensembles abstraits et leur application aux quations intgrales. Fund. Math. 3, 133-181.

[2] Barnsley, M. and Sloan, A. (1986). A better way to compress images. Byte 13 (1), 215-223.

[3] Chen, J., Monga, R., Bengio, S., and Jozefowicz, R. (2016). Revisiting distributed synchronous SGD. arXiv preprint arXiv:1604.00981.

[4] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M. et. al. . (2012). Large scale distributed deep networks. In Advances in neural information processing systems (pp. 1223-1231).

[5] Black A., and Kokorin V. Distributed Deep Learning, Part 1: An Introduction to Distributed Training of Neural Networks. Skymind. 2016.

[6] Dong, L., Lv, N., Zhang, Q., Xie, S., He, L. and Mao, M. (2016). A Distributed Deep Representation Learning Model for Big Image Data Classification. arXiv preprint arXiv:1607.00501.

[7] Holland, J. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press Cambridge, MA, USA.

[8] Kargupta, H. Chan, P. (2000). Advances in Distributed and Parallel Knowledge Discovery. MIT Press Cambridge, MA, USA.

[9] Rumelhart, D. E. and McClelland, J. (1986). Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press Cambridge, MA, USA

[10] Simon H.A. (1962) The Architecture of Complexity. In: Facets of Systems Science. International Federation for Systems Research International Series on Systems Science and Engineering, vol 7. Springer, Boston, MA

[11] Strom, N. (2015). Scalable distributed dnn training using commodity gpu cloud computing. In Sixteenth Annual Conference of the International Speech Communication Association.

[12] Zhang, W., Gupta, S., Lian, X. and Liu, J. (2015). Staleness-aware async-sgd for distributed deep learning. arXiv preprint arXiv:1511.05950.