An Introduction to Cloud K-SVD

Saketh Aleti, Augustus Chang, and Parth Parikh

***Abstract*—With the constraints of big data, data science and analytics must be performed in novel and distributed ways. cloud K-SVD [1] is a decentralized dictionary learning algorithm based on K-SVD [2]. In this report we discuss the foundations behind cloud K-SVD: average consensus and dictionary learning.**

I. Introduction

The overall goal of the year-long project is to implement cloud K-SVD and machine learning on a distributed testbed and detect issues that arise in practice. Before realizing this goal, it is important to establish a deeper understanding of cloud K-SVD, and the algorithms that comprise it. As such, we have spent our time so far trying to simulate cloud K-SVD in MATLAB, building from the bottom up. Next, we will briefly review the algorithms used within cloud K-SVD and how we built upon them to finally be able to run trials on the MNIST dataset [3].

II. Background

A. Average Consensus

In many real world scenarios, security and cost constraints prevent the luxury of a centralized data site. Realistically, multiple data sites will be connected in some fashion. A consensus algorithm can enable these sites to reach a solution without explicitly communicating entire sets of data.

We used the distributed average consensus method detailed in [4], along with Metropolis-Hastings weights. This algorithm was performed on a complete, undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N} = \{1, 2, \ldots, N\}$ represents each of the N nodes, and $\mathcal{E}$ represents edges. Because each node has access to its own data, $(i,i) \in \mathcal{E}$, and connections are represented by $(i,j) \in \mathcal{E}$.

B. Distributed Power Method

The power iteration is a method used to find the principal eigenvector, $\vec{q}$, of a square, positive semi-definite matrix $\mathbf{M}$. The power method can be described as $\hat{q}^{(t)} = M\hat{q}^{(t-1)}$ where $\hat{q}^{(t)}$, the estimate of the dominant eigenvector, approaches $\vec{q}$ monotonically for each iteration t; the initial value of q is denoted as q^{init} and may be any non-zero vector.

A distributed version of the power iteration method involves using consensus averaging across all sites. All sites begin with the same starting vector, $\hat{q}^{init}$, and a matrix $\mathbf{M}_i$ [1]. At each iteration of the power method, the nodes come to a consensus on an estimate of $\vec{q}$ for the matrix $\mathbf{M}$ where

$$\mathbf{M} = \sum_{i=1}^{N} \mathbf{M}_i.$$

and N represents the total number of nodes. Applying the power method to $\mathbf{M}$, we can observe the following:

$$\hat{q}^{(t)} = M\hat{q}^{(t-1)} = \left(\sum_{i=1}^{N} M_i\right)\hat{q}^{(t-1)} = \sum_{i=1}^{N} M_i \hat{q}_i^{(t-1)}.$$

This algorithm achieves the same goal, but defines a $\hat{q}_i^{(t-1)}$ for each node; this vector is theoretically the same for every node, but contains some error since consensus averaging is used to find the last summation of the above equation. Because the output $\hat{q}^{(t)}$ requires both consensus averaging and the power method, the local estimate is affected by error from both algorithms. With enough consensus and power iterations, each site's estimate of the principal eigenvector will be significantly close to the actual principal eigenvector for $\mathbf{M}$ [5].

The following subsections aim to introduce common variables used in both the original papers and remainder of this report.

C. Dictionary Learning

The goal of dictionary learning is to learn an over-complete dictionary matrix $\mathbf{D} \in \mathbb{R}^{n \times K}$ and a coefficient matrix $\mathbf{X} \in \mathbb{R}^{K \times S}$ to represent a matrix of signals $\mathbf{Y} \in \mathbb{R}^{n \times S}$, where n is the dimension of the signals, K is the number of dictionary atoms, and S is the number of signals. Because the dictionary is over-complete, $n < K$, and D is a full-rank matrix, we may find an approximation such that:

$$Y \approx DX.$$

Thus, we can define the sparsest representation as a solution of

$$\min_{x} \|x\|_0 \quad subject \quad to \quad \|Y - DX\|_2 \leq \varepsilon$$

where ε denotes a parameter for the largest representation error permitted and $\|\cdot\|_0$ denotes the zero-norm: the number of non-zero entries of a vector. In regards to sparsity, the representation of each signal s may be limited to a linear combination of a certain number of dictionary atoms; this number is the sparsity and is denoted as T_0. Minimizing the error and restricting the sparsity would require us to solve the following:

$$\min_{D,X} \left\{\|Y - DX\|_F^2\right\} \quad subject \quad to \quad \forall s,\ \|x_s\|_0 \leq T_0.$$

D. K-SVD

We can solve the aforementioned minimization problem by using K-SVD [2], an algorithm that approximates a solution by determining a sparse representation with a given dictionary and then improving upon that representation. K-SVD repeats this two stage process until convergence or a stopping rule is reached.

The first stage is defined as the Sparse Coding Stage, where a pursuit algorithm is used to compute the representation vectors x_s to approximate the solution of:

$$\forall s, \quad \min_{x_s} \left\{ \|y_s - Dx_s\|_F^2 \right\} \quad s.t. \quad \|x_s\|_0 \leq T_0.$$

The second step is defined as the Codebook Update Stage, where each dictionary atom, $D^k \ \forall \ k = 1, 2, \ldots, K$, is updated by the following process, beginning with the creation of a group of indices:

$$w_k = \left\{ s \,|\, 1 \leq s \leq S, \, x_T^k(s) \neq 0 \right\}$$

where w_k is a row vector that represents the indices of group of signals that use this atom, while x_T^k is defined as the k^{th} row of x. Then we find the overall representation error matrix, E_k by finding:

$$E_k = Y - \sum_{j \neq k} d_j x_T^j.$$

We then restrict E_k to only those columns that correspond to w_k, which is essentially removing consideration for error that $d_k x_T^k$ is not responsible for; selecting those corresponding indices results in E_k^R. The goal is to then solve the following:

$$\min_{D,X} \left\{ \|E_k^R - d_k x_R^k\|_F^2 \right\}.$$

In order to do this, we must find two vectors to represent d_k and x_R^k that are able to solve the minimization problem. This can be solved by finding a low-rank approximation of E_k^R [6]. This is done by first doing the singular value decomposition (SVD) of E_k^R such that:

$$E_k^R = U \Delta V^T$$

where U is an orthogonal matrix, Δ is a diagonal matrix composed of the singular values (square roots of the eigenvalues) of E_k^R from largest to smallest, and V is another orthogonal matrix composed of the eigenvectors of E_k^R corresponding to each singular value in Δ. These matrices can be broken down into column vectors which sum into E_k^R; that is,

$$\sum_i U_i \Delta_{(i,i)} V_i^T.$$

We may then construct a rank-one approximation [6] by selecting the vectors of the decomposition corresponding to $i = 1$. The vectors correspond to the singular values found in Δ, which is ordered from greatest to least, so $i = 1$ is the largest. Since U and V are both orthonormal, the magnitude of the output matrix is determined by the scalar Δ_i. This means we capture the matrix in the summation with the greatest significance or "energy;" this is best two vector representation of E_k^R. We may then assign our atom and coefficient vector these values, which are the solution to the minimization problem for E_k^R.

$$d_k = U_1 \qquad x_R^k = \Delta_{(1,1)} V_1$$

We can then transform x_R^k back into x_T^k by setting the indices of x_T^k corresponding to w_k as the values of x_R^k and all other values as zero. This essentially sets $d_k x_T^k$ as rank-one approximation of E_k^R thereby minimizing the error of the approximation of the signals Y. This is done for all atoms of the dictionary in one iteration. After this stage, the iteration count is updated, and this two-step process repeats until convergence is reached [2].

E. Cloud K-SVD

We can apply the K-SVD method for dictionary learning to a distributed setting where local data is held in different sites. Each site is denoted by S_i with local data $Y_i \in \mathbb{R}^{n \times S_i}$ for a total of N sites; thus, the total number of data samples is $S = \sum_{i=1}^N S_i$, and all the data can be expressed as one matrix $Y \in \mathbb{R}^{n \times S}$. The goal of cloud K-SVD is for each site to collaboratively learn a dictionary D that solves the same minimization problem as K-SVD.

In order to do this, we first generate a common random reference vector $d^{ref} \in \mathbb{R}^n$ at every site along with an initial random dictionary, $D^{init} \in \mathbb{R}^{n \times K}$. We define this as the first local dictionary estimate, denoted as $\hat{D}_i^{(t-1)} \ \forall i$ where t denotes the iteration of the cloud-KSVD algorithm, for $t = 0$. Since i is defined from $1, \ldots, N$, $\forall i$ means the same as for each node. With these initial parameters, we move on to the main algorithm, which contains a Sparse Coding stage and a Dictionary Update stage like centralized K-SVD.

In the Sparse Coding stage, we aim to solve the following problem at each site:

$$arg \min_{x_s} \|y_{i,s} - \hat{D}_i^{(t-1)}\|_2^2 \quad subject \quad to \quad \|x\|_0 \leq T_0$$

We can locally compute, using a pursuit algorithm, a sparse representation of Y_i using $\hat{D}_i^{(t-1)}$ in each individual site; in the MATLAB simulation, we chose batch-OMP described in [7] as our pursuit algorithm. Following, we elaborate on some terms related to batch-OMP. Matching pursuit, MP, is an algorithm used to decompose a signal into linear combinations, or an approximation, of the columns of some dictionary [8]. OMP, orthogonal matching pursuit, is an improvement on MP in that it is guaranteed to converge in no more than N steps, where N is the number of dictionary columns. The OMP algorithm is described in [9]. We chose batch-OMP, an implementation of OMP, because it is particularly suited for K-SVD, that is, for sparse-coding large sets of signals [7].

In the Dictionary Update stage, we aim to repeat the same steps of K-SVD, but iteratively to achieve a collaborative dictionary. We may recall that we defined each dictionary atom, $d_k \ where \ k = 1, 2, \ldots, K$, in K-SVD. We apply a similar process in a distributed setting where we aim to achieve a collaborative dictionary. That is, our goal is to find D, a collaborative global dictionary. Because we approximate D, it varies slightly between each node n, so instead we find

$$\hat{D}_i^{(t)} \quad \forall\, i = 1, 2, \ldots, N$$

In order to do so, we update the dictionary atoms $d_{i,k}$ at each node by collaboratively finding the total error that atom must minimize at every node. This first requires us to find the local representation error matrix at each site: $\hat{E}_{i,k,R}^{(t)}$. This in turn requires the w_k to be calculated at each site:

$$w_{i,k}^{(t)} = \left\{ s \,|\, 1 \leq s \leq S_i,\, x_{i,k,T}^{(t)}(s) \neq 0 \right\} \forall i.$$

In this case, $w_{i,k}^{(t)}$ refers to the indices used by the dictionary atom $D^{i,k}$ at iteration t. Rather than using $w_{i,k}^{(t)}$ to select indices, we can define another matrix, $\Omega_{i,k}^{(t)}$, as an $S \times |w_{i,k}^{(t)}|$ binary matrix with ones in $(w_{i,k}^{(t)}(s), s)$. Thus, we may define $\hat{E}_{i,k,R}^{(t)} = \hat{E}_{i,k}^{(t)}\Omega_{i,k}^{(t)}$ to simplify the process; defining $\hat{E}_{i,k}^{(t)}$may be done with the following equation:

$$\hat{E}_{i,k}^{(t)} = Y_i - \sum_{j \neq k} \hat{d}_{i,j}^{(t)} \hat{x}_{i,j,T}^{(t)}.$$

where j is defined for $1, ..., K$. The next goal is to calculate the singular value decomposition of $\hat{E}_{k,R}^{(t)} = \sum_{i=1}^{N} \hat{E}_{i,k,R}^{(t)}$ in order to find $U_{1,k}$ and $V_{1,k}$, which were defined earlier for K-SVD. Again, we make the following updates:

$$\left(\hat{d}_{i,k}, \hat{x}_{i,k,R}\right) = \left(U_{1,k}, \Delta_{(1,1)} V_{1,k}\right) \quad \forall i.$$

However, by definition, we know that

$$\hat{x}_{i,k,R} = \hat{d}_{i,k}^{(t)^T} \hat{E}_{i,k,R}^{(t)} = U_{1,k} \hat{E}_{i,k,R}^{(t)} = \Delta_{(1,1)} V_{1,k}. \quad \forall\, i$$

In order to find a rank-one approximation of $\hat{E}_{i,k,R}^{(t)}$, we may use the distributed power method to find its dominant eigenvector. This requires $\hat{E}_{k,R}^{(t)}$ to be a square, positive semi-definite matrix, which is not necessarily true in many cases. Instead, we may apply the distributed power method to $\hat{E}_{k,R}^{(t)} \hat{E}_{k,R}^{(t)^T}$; by multiplying the matrix by its transpose, we are guaranteed to produce a positive semi-definite matrix. We can nominally define this matrix as:

$$\hat{M_k}^{(t)} = \sum_{i=1}^{N} \hat{M}_{i,k}^{(t)} = \sum_{i=1}^{N} \hat{E}_{i,k,R}^{(t)} \hat{E}_{i,k,R}^{(t)^T}.$$

When we apply the distributed power method, we find the dominant eigenvector of $\hat{M}_k^{(t)}$; by definition, the dominant eigenvector of this matrix is the U_1 of $\hat{E}_{k,R}^{(t)}$, since the left singular vectors of any matrix are the eigenvectors of that matrix multiplied by its transpose. Knowing this, we can then set $\hat{d}_{i,k}$ to the result of the distributed power method. We may then locally compute $\hat{d}_{i,k}^{(t)^T} \hat{E}_{i,k,R}^{(t)}$ to set $\hat{x}_{i,k,R}$. Lastly, we set $\hat{x}_{i,k} = \hat{x}_{i,k,R} \Omega_{i,k}^{(t)^T}$ to place the zeros back into the row of coefficients. This completes the calculation of a dictionary update for one atom; updating every atom in the global dictionary completes the Codebook Update stage. The entire loop then repeats for each dictionary learning iteration (t).

III. Simulation and Results

The MNIST data set [3] was used to evaluate the practical efficacy of cloud K-SVD. The dataset was first analyzed by centralized K-SVD and the error of the results were compared to that of cloud K-SVD; for centralized K-SVD, the method is as follows.

The images of the digits {0,3,5,8,9} were extracted from the training dataset and used for dictionary learning; we define a class $c = 1, .., 5$ for the 5 types of digits. Specifically, this results in a dictionary $D = [D_1 \;\; D_2 \;\; D_3 \;\; D_4 \;\; D_5]$ where D_c corresponds to the respective subset of data from one digit. Each atom in the dictionary corresponds to one image from its respective class. The signal matrix, Y, was defined in a similar way where each signal was a random image from the MNIST test sample dataset. By solving the dictionary learning problem

$$\min_{D,X} \left\{ \|Y - DX\|_F^2 \right\} \quad subject \quad to \quad \forall s,\, \|x_s\|_0 \leq T_0$$

we aim to find what training images can produce the test images with the least error. The coefficient matrix X was partitioned as well, since it essentially produces linear combinations of the training images to produce the test samples. We partition it similarly to D where X_c is composed of the rows of X that correspond to the atoms D_c. Therefore, $D_c X_c$ would be an approximation of the signals Y but only composed of the training images corresponding to class c. Intuitively, we define the class specific approximation for some signal s as $D_c x_{s,c}$; the error between this representation and the actual signal is the residue for class c. So, having defined residue as, $r_c = \|y_s - D_c x_{s,c}\|_2$, we define the detected class $c* = arg\, min_c r_c$. The overall performance is then defined as $R_c = \frac{samples\ in\ c\ detected\ correctly}{total\ samples\ in\ c}$ [1].

Determining the error of cloud K-SVD involves the exact same method. In cloud K-SVD, we aim to find a collaborative, global dictionary D. This results in marginally more classification error than K-SVD due to error from consensus averaging and the distributed power method.

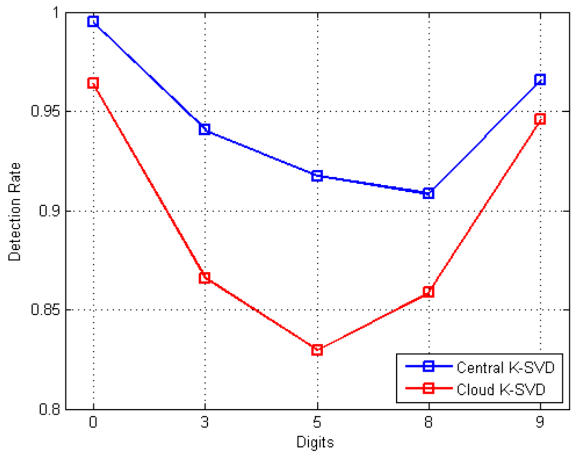
After running cloud K-SVD, we find the residue locally in the same way by calculating residue for each node as well as each class. This is defined as $r_c = \|y_s - D_c X_{s,c}\|_2$. After a global dictionary is achieved, the classification error can be found at each node and then averaged for each class; this allows us to compare the efficacy of a global dictionary to a local one. The results of a simulation with the following parameters is graphed below.

Simulation Parameters[1]

- Image Scaling = 0.3265
- Global Training Samples = 1000

[1]The number of global/local training samples and test samples denoted are per digit.

- Nodes = 10
- Local Training Samples = 100
- Test Samples = 500
- Iterations of cloud K-SVD = 7
- Iterations of Distributed Power Method = 10
- Iterations of Consensus Averaging = 10
- Sparsity, T_0 = 10



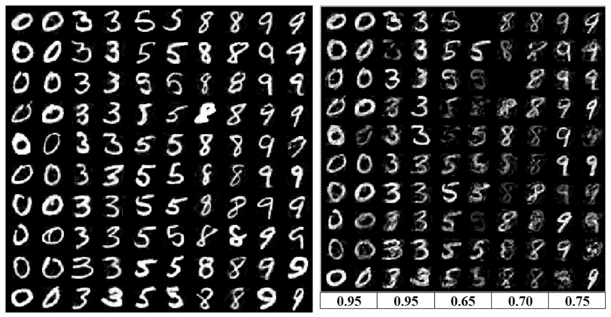
The results above are the average of 100 Monte Carlo trials. In each trial, the training samples and test samples are chosen at random, independent of any previous trials. The average detection rate for every class by K-SVD was 94.54%, while that of cloud K-SVD was 89.28%.

IV. Discussion

A. Intuition Behind MNIST Dictionary Learning

The procedure we used to apply dictionary learning through K-SVD to the classification problem represented by the MNIST test image database was essentially supervised machine learning. There exist several ways to approach the problem itself using simply dictionary learning with supervised being a method resulting in relatively low error [10]. In any case, the justification behind the application of dictionary learning was based on the theory that the sum of the residuals of the representation of test images of class c would be lowest when represented by dictionary atoms corresponding to images of the same class c. In other words, test images of the digit 0, for example, would best be represented by linear combinations of the training images of the digit 0; better representation results in lower residuals. The idea of using known classifications of dictionary atoms to predict unknown test samples would then be the previous argument in reverse.

Because K-SVD aims to reduce error at every stage, this argument may not always be true since error is reduced, but the residuals may not necessarily change to better represent the classification of the digit. For example, Figure 1 (a) above shows the representation of 100 test samples after 10 iterations of centralized K-SVD with $T_0 = 5$. The second figure shows the same representation, but only using atoms that are of the same class as the sample images. Listed below each set of digits is the detection rate for each class of digit. Note how representations that primarily use linear combinations of training images in the same class as the test image they aim to represent have better detection rates. Even though the error in the representations for digits 0 and 8 may be very low and similar (the average representation error for both were 0.031 and 0.022 respectively), the class of the atoms that are used in their representation are primarily responsible for whether they are detected correctly or not. We can see in Figure 1 (b) that the classification accuracy of 0's was 95% compared to that of 8's, which was 70%, even though images of the digit 8 had lower representation error. This is the source of the error in detection rate that arises from this form of supervised machine learning for classification. One algorithm that does take this into consideration is D-KSVD, which minimizes errors resulting from misclassifications in its algorithm; as a result, it is able to perform better in classification problems than regular K-SVD [11].



(a) Sparse Representation (b) Class c Representation

Fig. 1: Representations of 100 unique handwritten digits found using K-SVD. (a) Each digit s is represented normally (by DX_s). (b) Each digit s is represented by $D_c x_{s,c}$ where c is the actual class of s.

B. Active Cloud Dictionary Learning

In practical application, we can expect that the implementation of cloud K-SVD will be justified by its ability to make column-wise updates that incorporate all other training samples. However, the addition of a new set of signals in a local setting, which may occur if new data is uploaded to a lone site, requires updates; these updates may require a significant amount of time for large data sets in a cloud setting due to the time taken by the distributed power method and OMP locally. In such a scenario, a previously developed algorithm, "Active Dictionary Learning," to actively screen a training set for column-wise K-SVD updates may be used [12].

Let $Y_{t,s} = [y_{i,t}, \ldots, y_{N,t}]$ denote a new set of training signals uploaded to site s, and $Y_{init,s}$ be the current set of signals at the same site. We would then locally compute

$$\hat{y}_{i,\,t} = D_{t-1} x_{i,t} \,\forall\, i$$

using OMP to generate a sparse representation. The representation error for these approximations are calculated as such: $\epsilon(y_{i,t}) = \|y_{i,t} - \hat{y}_{i,t}\|_2^2$. Signals with the largest representation error are those not represented well by the current dictionary D_{t-1}. Rather than update the cloud dictionary to fit the batch $Y_{t,s}$ in its entirety, we select a fraction of those samples, $y_{i,t}$, above a certain error and add those to the learning pool. This essentially reduces the size of $Y_{t+1,s}$, the new samples at site s, by only considering signals that would not adequately be represented by the current dictionary. Thus, the amount of computation and time required to update an appropriate cloud dictionary with a new batch of data would be reduced without significantly affecting the representation capability of the dictionary [12]. While this approach was previously described for centralized computations of K-SVD, it may be useful and worth pursuing in a cloud-based implementation.

V. Conclusion

This paper encompasses our semester's worth of research. We learned the foundations of cloud K-SVD: distributed methods and a centralized method of dictionary learning. The two concepts intersected when we learned how the dictionary update stage could be computed in a decentralized setting by using the distributed power method. Finally, we were able to apply cloud K-SVD to classify MNIST images on MATLAB. We look forward to implementing cloud K-SVD on a network of micro controllers in the following semester.

References

[1] H. Raja, W. Bajwa, "A Convergence Analysis of Distributed Dictionary Learning based on the K-SVD algorithm," *Information Theory (ISIT), 2015 IEEE International Symposium*, vol., no., pp.2186-2190, 14-19 June 2015

[2] Michal Aharon, Michael Elad, and Alfred Bruckstein. "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation." *IEEE Transactions on Signal Processing IEEE Trans. Signal Process.* (2006)

[3] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/, 1998

[4] L. Xiao, S. Boyd, S. J. Kim, "Distributed Average Consensus with Least-Mean-Square Deviation," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 33-46, 2007.

[5] M. Panju, "Iterative Methods for Computing Eigenvalues and Eigenvectors," *The Waterloo Mathematics Review*, vol. 1, pp. 9-18, 2011

[6] Eckart, Carl, and Gale Young. "The approximation of one matrix by another of lower rank," Psychometrika, vol. 1, no. 3, pp. 211-218, 1936.

[7] R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit. Technical Report - CS Technion, April 2008.

[8] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries", IEEE Trans. Signal Process., vol. 41, no. 12, pp. 3397 -3415, 1993

[9] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," *1993 Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pp. 40-44, 1993.

[10] T. Bufford, Y. Chen,M. Horning,L. Shee. "When Dictionary Learning Meets Classification," *University of California, Los Angeles*, 2013.

[11] Z. Jiang, Z. Lin, and L. Davis. Learning a Discriminative Dictionary for Sparse Coding via Label Consistent K-SVD. In *CVPR*, 2011.

[12] T. Wu, A. Sarwate, and W. Bajwa,"Active dictionary learning for image representation", *Unmanned Systems Technology XVII*, 2015.