EE 510: Mathematical Foundations of Machine Learning **Winter 2021**

Course Project: Dictionary Learning
Due: March 19, 2021, 11:59PM PT

Student Names: Anthony Hosti, Andrew Forsman, Mikhail Mayers

1 Paper Descriptions

1.1 Paper 1

Paper Title: Complete Dictionary Learning via ℓ^4-Norm Maximization over the Orthogonal Group
Student Name: Anthony Hosti

1.1.1 Problem Description & Formulation

If you're not familiar with the show "The Blacklist" I would highly recommend checking it out. Avoiding an in-depth discussion of its premise, the show has an episode in which two programmers are working to break into a banking computer system for which they lack a password of the intended users account they wish to hack (what follows is a quick paraphrase of the main ideas discussed in the scene). The second hacker says to the first, "Oh, you're brute forcing it?" to which the first hacker replies, "Of course, how else are we supposed to generate a password?", as he looks up to find the second hacker running a scan of their target's social media and the first hacker comments, "Are you scanning their social media? There aren't going to be enough data points to generate a usable dictionary to propose a viable password."

Now, the first time I saw this show I did not have the faintest idea what "dictionary" they were talking about. As it turns out, the scene is a perfect backdrop for the explorations of this paper. The primary goal of the authors of this paper is to devise a more efficient system for recovering usable dictionaries and their associated sparse data sets from the output data. So, in the case of "The Blacklist" the intended dictionary provided a means for making educated guesses (the dictionary) of the user's banking password (the sparse data) from public records and social media (the output data). But the tools developed in this paper can be used to generate just about any dictionary a user might need as well as the original sparse data set from the output data and it does so at a remarkably faster pace than many of the other most popular algorithms of the day.

But how are these ends achieved? The authors rely heavily on theory to build their algorithm, specifically the theory surrounding the ℓ_4 norm. As the authors lay it out, the ℓ_4 norm, for various reasons, ends up being the ultimate maximizer for rebuilding dictionaries from output data. The authors lay out four specific fields of study that informed their used of the $\ell_4 norm$

- **Spherical Harmonic Analysis:** study here has shown that ℓ_4 norm is locally maximized by the highest weight function with the result that it is the most concentrated in measure along the unit sphere.
- **Independent Component Analysis:** study here has shown that if x_i is independent and identically distributed Bernoulli-Gaussian then the goal of ICA is the same as maximizing the ℓ_4 norm of a vector over a sphere.
- **Sum of Squares:** study here shows when $\mathbf{x}$ is sufficiently sparse it is possible to recover a dictionary from a higher order sum of squares like the ℓ_4 norm.
- **Blind Deconvolution:** study here shows that the geometry of the ℓ_4 norm over a sphere ensures that saddle points have a negative curvature making them easy to identify and cancel out. [1]

Using this underlying theory they engineer an algorithm focused on the efficient return of a dictionary prioritizing speed over the error between the data sets. Despite prioritizing speed over accuracy, it turns out the error also remains very low and in the neighborhood of other similarly focused algorithms.

The authors acknowledge that their algorithm is still a work in progress as it still lacks a rigorous proof; but the empirical data they generate is very promising. Especially when you consider that other popular methods are only able to generate a usable dictionary a single column at a time while the algorithm proposed in this paper generates a complete dictionary all at the same time. That makes this algorithm incredibly powerful.

1.1.2 Algorithm Description

The algorithm in this paper utilizes the ℓ_4-norm to maximize an optimization over the entire orthogonal group, which is "is the group of distance-preserving transformations of a Euclidean space"[2]. And as I mentioned above, it generates a complete dictionary holistically as opposed to one column at a time using the conventional format $Y = D_0X_0$ where Y is the output from which you derive D_0, the "ground truth dictionary" and X_0 the associated sparse data matrix. So, in this paper, the authors work backwards from the output signal to recover the dictionary and the data set that generated the observed output. Other than the ℓ_4-norm, the cost of this algorithm is a simple SVD. The SVD is then utilized to perform simple iterative matching, stretching, and projection operators which is why the authors refer to their algorithm as the MSP algorithm. The output of this algorithm is the desired dictionary in the form of a signed permutation matrix which can be adapted to recover the original data set, X_0.

Another of the characteristics that drew the authors to the ℓ_4-norm is the fact that the ℓ_4-norm is smooth all around unlike the ℓ_1-norm which is a popular norm used in similar algorithms. The smoothness of the ℓ_4-norm makes the optimization simpler. As well, they find in their experiments that the ℓ_4-norm seems to be the best in terms of trade-offs as it simultaneously keeps the size of the data necessary to generate the desired dictionary small while also achieving the desired speed and error rates.

The empirical results presented are astonishing. And the authors even provide a comparison of the performance and run times of their algorithm against the KSVD, SPAMS, and Subgradient descent, which are the most common algorithms in this subgroup. Based on the data, their algorithm grossly outstrips the competition in speed with little sacrifice in error. SPAMS and Subgradient descent provide less error, but they do so in substantially more time. With a sample size of 400 and up to a ℓ_p-norm of 16, the MSP algorithm delivers an error of .35% and a computation time of 8.24 minutes to the almost 12 hours of the Subgradient descent or SPAMS. The error rate for the competition isn't provided for this specific data point.

1.1.3 Theoretical Results

The primary assumption that underlies the work in this paper is that the matrix X_0 is independent and identically distributed random variables which is a complicated way to say that the matrix columns are linearly independent and orthogonal. As mentioned previously the paper still lacks a rigorous proof but their results thus far are very promising. Thus far they have data that suggests this algorithm can work with any norm greater than 2 but with trade-offs. The larger the norm used the more data becomes required to generate the holistic dictionary, hence one of the reasons the authors make the decision to use the ℓ_4-norm.

As well, the algorithm relies on convergence to a maximum that is achieved more quickly the larger the norm used, which seems obvious and almost seems to miss the point as they show in the paper that a norm of $+\infty$ results in the fastest maximization. After all, what can lead you to a maximum faster than multiplication by $+\infty$, right?

The rigorous proof of guaranteed optimality and convergence the authors leave for future work.

1.1.4 Relation to Course Material

This article has significant overlap with the material covered in this course. In this paper the authors compares their work to the KSVD, about which we have read the same article twice. It also utilizes the

SVD and different norms in a similar fashion to the way we have used them in homework and on tests. As well, the authors test their algorithms against the MNIST data set which we have used several times in homework and demos. And the algorithm itself is an optimization problem like all the optimization problems we have studied this term. I must say, I feel significantly more grounded in the material presented in this paper now than I would have been before taking this course. Without a doubt I would have had little to no understanding of most of the topics presented in this paper prior to taking this course. I still feel very disconnected from the actual implementation steps for these algorithms. I am still learning how to connect the theoretical descriptions of the algorithms to the actual code used to implement them. My brain is still acclimating to how iterative loops are coded. I still struggle with the fact that you don't seem to need to directly layout the indices for the loops.

And last but certainly not least, I still feel overwhelmed when reading the larger more complex proofs like those found in this paper. To give a point of reference, in reading about Gauss, the way he described his own work; he preferred to show the building without all of the scaffolding. I'm still in the phase with analysis that I need and desire to see the scaffolding as much as the building. But unfortunately, thanks in part to Gauss, analytical mathematics is still lacking the foundational "scaffolding descriptors" that to me seem a requirement for this level of analysis. Imagine how many more students would love mathematical analysis if there were something as simple as a dictionary of mathematical terms and symbols to which one could refer when they encounter a symbol or character whose use is unfamiliar. These things can be highly distracting.

You would not study a language without a dictionary of terms. Why do we study math without one? Ah, that question is for another day.

1.2 Paper 2

Paper Title: Trainlets: Dictionary Learning in High Dimensions
Student Name: Mikhail Mayers

1.2.1 Problem Description & Formulation

Dictionary learning a sparse coding method used to update atoms in the atoms of a dictionary. It's been successfully used in compression, denoising, and inpainting, to name a few. The dictionary itself is a learned group of features. The columns of a dictionary are called atoms, and the atoms are selected by a representation vector. A representation vector is a sparse input that selects only the necessary features of the dictionary. The atoms themselves can be thought of as "features" of the signal. So the typical way of writing this is:

$$y = Dx$$

Where $D \in \mathbb{R}^{N \times M}$ is the dictionary, $x \in \mathbb{R}^M$ is the representation vector and $y \in \mathbb{R}^N$ is the observed signal. We solve the sparse representation problem as an optimization problem by using:

$$\min_x \|x\|_0 ; \quad \text{where: } \|y - Dx\|_2 \leq \xi$$

To explain this equation in layman's terms, the x with the least amount of zeros. Or more specifically, what features from the dictionary best describe the signal, using the smallest number of features.

In the case of images, dictionary learning performs best with images, much smaller than 64x64. Tackling this problem usually involves breaking an image into smaller piecess, learning each piece, then putting the pieces back together into the whole image. However, a major issue are the borders of the reassembled pieces. The borders tend to be incomplete or distorted because they are filled in using less than optimal method, such as an average. And when taking a bunch of small averages that will later be reconstructed into a whole errors are inherent in the reconstruction. For this reason, other methods for computing large dimensional data and large images often end up being computationally inefficient and therefore unusable.

The method of *Trainlets* works to overcome these problems. The two big methods introduced in this paper involve *separable cropped wavelets* to build the dictionary, which are a variation on wavelets and the use of an algorithm called *Online Sparse Dictionary Learning (OSDL)* to train the dictionary.

1.2.2 Algorithm Description

Though this is not a step in the algorithm, it is important to understand that rather than using the typical dictionary D, this paper uses a separable dictionary $D = \Phi A$, where Φ is a fixed operator. The idea is that Φ will be a base dictionary, and A is a sparse adaptable matrix. Together they make the dictionary, but only A is updated by the algorithm. This form was chosen because Φ, the base dictionary of separable cropped wavelets converges faster, and updating only A in the algorithm is more efficient. The expressed separable cropped wavelet is itself a dictionary. The cropped wavelet is defined as

$$\Phi_1^c = P^T W_s \mathbb{W}$$

where, $P \in \mathbb{R}^{L \times n}$ is the zero padding, $W_s \in \mathbb{R}^{L \times L}$ is the Wavelet Synthesis Operator, and $\mathbb{W} \in \mathbb{R}^{L \times L}$ is a diagonal matrix to give Phi_1^c unit $norm$[3]. To make Phi_1^c separable, the Kronecker product

$$\Phi_2 = \Phi_1 \otimes \Phi_1$$

is used to make a completely separable sparse Wavelet representation.

The paper states the algorithm is "a block coordinate minimization over this non-convex problem", where the algorithm updates the representation vector at one point by fixing the dictionary, then switches and updates the dictionary and fixes the representation vector. This means that the algorithm switches between doing the sparse coding on the signal example Y and then switches to update the atoms in the dictionary simultaneously.

The initial parameters are Y the training samples, Φ the Base Dictionary and an initial sparse matrix A. We initialize the the gram matrix $G = \Phi^T \Phi$ and set $U = 0$. The Gram matrix makes the update of the atoms in the dictionary less computationally expensive, and U is a "momentum" term, set to speed up convergence. From here the algorithm iterates for T times.

$$\textbf{for } t = 1, \ldots, T \textbf{ do}$$

The algorithm OSDL now starts by grabbing a random mini-batch of Y called y_t.

$$y_t \in Y$$

We then start the sparse coding step to solve for X_t.

$$X_t = Sparse(y_t, \Phi, A^t, G^t) = \min X \frac{1}{2} \left\| y_t - G^t A^t X_t \right\|_F^2$$

We then solve for η which is the gradient descent step size. Using η we solve for a new U which speeds up convergence.

$$\eta^t = norm f(A_S^t)_F / \left\| \Phi f(A_S^t) X_t^S \right\|_2$$

$$U_S^{t+1} = \gamma U_S^t + \eta^t f(A_S^t)$$

The next step is updating the atoms in A, and then using A to update the gram matrix.

$$A_S^{t+1} = P_k[A_S^t - U^{t+1_S}]$$

$$G \Leftarrow (A^{t+1})^T G_\Phi A_S^{t+1}$$

By using the separable dictionary $D = \Phi A$, updating the atoms becomes computationally more efficient. Because A and X both act as sparse vectors, we call this the *Double Sparsity model*.

1.2.3 Theoretical Results

As patch size and sparsity increase the double sparsity model shows its advantage in it's ability to minimize errors along the borders of the image. This reduces of error overall as the patch sizes increase with the added benefit that it also reduces the computational complexity in high dimensions.

The learning process still needs to be efficient in order to make use of the double sparsity model. Separable cropped wavelets alone are not enough to get a proper results. OSDL was created to maximize the effectiveness of the separable cropped wavelets, and without a proper base dictionary can not achieve its desired results.

1.2.4 Relation to Course Material

This paper mentions the following topics that relate to the class:

- stochastic gradient descent
- optimization, convergence and non-convex problems
- minimization using the Frobenius norm
- basis pursuit
- batch processing
- K-SVD
- Gram matrix
- PCA

Some of the aspects I do not quite understand are wavelets. Haar wavelets are something I was interested in learning, but I think they are out of scope for this class. A lot of the other topics I was confused on were some of the other algorithms they talked about, like Sparse K-SVD and NHDL, though it's discussed in this paper.

1.3 Paper 3

Paper Title: Task-Driven Dictionary Learning
Student Name: Andrew Forsman

1.3.1 Problem Description & Formulation

Dictionary learning is a technique that involves decomposing a signal into linearly combined bases learned from the data. This can be used for things like representing high-dimensional data in lower dimensions. These representations can then be used for recovering the original data. Unsupervised (or Data-Driven) Dictionary Learning involves performing this process on data that isn't labeled, or in other words contains unknown features, and has been very successful at data reconstruction. This involves recreating the original signal from a noisy, sparse, or otherwise imperfect sample of that signal. This works by creating dictionaries that are very similar to dictionaries created from the original signal and from which the original data can be reconstructed. There are other tasks (like classification and regression) that have been shown to be more successful when using supervised learning (training using labeled data). This paper describes a general formulation for learning dictionaries that are suited for those previously mentioned tasks using supervised or semi-supervised learning, as well as an efficient solution to the optimization problem that falls out of the generalized formulation.

A difference between these two approaches is that with unsupervised dictionary learning, optimization methods are known and proven, whereas with the supervised or task-driven dictionary learning, optimization

has been a more difficult problem. This paper proposes a solution to that problem. The general formulation involves minimizing this function over **D** and **W**, the dictionary and the parameters respectively:

$$\mathbb{E}_{y,x}[\ell_s(\mathbf{y}, \mathbf{W}, \boldsymbol{\alpha}^*(\mathbf{x}, \mathbf{D}))] + \frac{v}{2} \|\mathbf{W}\|_F^2$$

in order to learn **D**, the dictionary, and **W**, the model parameters for the task, simultaneously. In this equation v is a regularization parameter. This optimization works because we are trying to minimize the convex loss function ($\ell_s(\cdot)$). The lower the value of the loss function, the closer the predicted **y** is to the actual **y**, so if we minimize the function we will have produced the dictionary and parameters that give the smallest difference between the predicted and actual **y**.

1.3.2 Algorithm Description

This algorithm uses an iterative optimization method, Stochastic Gradient Descent, which is something we have used this term. The algorithm uses this method to minimize the elastic-net loss function with respect to D, the which is the dictionary matrix. The elastic-net loss function is similar to the Lasso (which we also used this term) but includes a second regularization term, $\frac{\lambda_2}{2} \|\cdot\|_2^2$. In fact, the Lasso is a special case of the elastic-formulation, where $\lambda_2 = 0$. The regularization parameters ensure that penalties for α being large are taken into account when minimizing the overall loss function. When $\lambda_2 > 0$, this formulation is strongly convex, and so SGD should converge on a minimum for **D**.

The method is to first learn a dictionary $\boldsymbol{D}$ in the classic unsupervised way, by minimizing the expected value of a loss function with respect to $\boldsymbol{D}$. Next, that learned dictionary is used to minimize the convex cost function

$$\arg\min_{\boldsymbol{W}} f(\boldsymbol{W}) + \frac{v}{2} \|\boldsymbol{W}\|_F^2$$

where **W** are the model parameters we wish to learn in order to accurately predict **y**. This gives us the initial dictionary and parameters for use in the algorithm.

The first step in the algorithm is to set initial parameters for $\lambda_1,\ \lambda_2$ which are real numbers, the matrix **D**, which is the initial dictionary, the matrix **W**, which contains model parameters we want to learn, T, the number of iterations, and $t_0,\ \rho$, which are learning rate parameters.

This is achieved by first taking some random sample from the training data (a feature and a label) and then minimizing the following elastic-net loss function using Least Angle Regression:

$$\boldsymbol{\alpha}^* = \arg\min_{\alpha \in \mathbb{R}^P} \frac{1}{2} \|x_t - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda_1 \|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\alpha}\|_2^2$$

This will create a sparse vector to be multiplied by the dictionary, which will result in a sparse representation of x_t. Next, find where $\boldsymbol{\alpha}^*$ is not equal to zero, and then use those indices to calculate a vector appears in the sub gradients with respect to **W**, and **D** of the task-driven learning equation. Choose the minimum of the learning rate ρ or $\rho\frac{t_0}{t}$ and then update the parameters and the dictionary (**W,D**) by using a projected gradient step, and then restart the process by taking a random sample from the training data. The process ends when the number of iterations reaches T and at that point the dictionary has been learned.

This works because first the vector $\boldsymbol{\alpha}$ is calculated to minimize the error and then the dictionary and parameters take a "step" towards their minimum, and are updated in regards to that new $\boldsymbol{\alpha}$. Then another input/output pair is inserted and $\boldsymbol{\alpha}$ is updated again. While this does not result in a monotonic path to minimization (because the input/output vectors are not in any particular order, so it is possible that the current minimization is worse overall than the previous one), with enough iterations, the solution will converge to something that is optimized, and so will the dictionary.

This works well for task based learning because we are minimizing **W** and **D** with the end result in mind, whether that be classification, regression, or some other task like learning a linear transformation. Where standard dictionary learning seeks to learn a dictionary to be used to reconstruct a signal, this learns a dictionary, and model parameters, that are used to predict an output based on an input.

The algorithm was tested on classification and regression tasks, involving the classification of handwritten digits, reconstruction of halftone images, and authentication of artwork.

In testing the algorithm for handwritten digits classification on the MNIST data set, the algorithm achieved "state-of-the-art results", with an error rate of 0.54%. On the USPS dataset, it achieved an error rate of 2.84% which is slightly higher than the rate of 2.4% given by another algorithm. For dictionaries of the same size, the algorithm's errors were significantly lower than one created with unsupervised learning.

When applied to the task of regression in the form of reconstructing the original image from a half-tone transformation, the algorithm performed better or equal to the previously state-of-the-art algorithm's results in 11 out of 12 sets.

Using the algorithm for classification of authentic vs. imitation art, it performed slightly better than the unsupervised case, with an succes rate of 55.04% compared to 51.94% for individual image patches of the art, and correctly identified an imitation in all tests, whereas the unsupervised algorithm failed one out of eight tests.

Finally, the algorithm was tested on a reconstruction task of dimensionally reduced data (this is the compressed sensing task). When initializing the matrix that performs the dimensionality reduction ($\mathbf{Z}$), when it is initialized using PCA and then found via supervised learning, the algorithm performs significantly better than any other combination of techniques using random or PCA initialization, and unsupervised or supervised learning.

1.3.3 Theoretical Results

This paper makes three assumptions that must be true for the optimization algorithm to work:

1. The input/output data admits a probability density, p, with a compact support, $K_{\mathcal{Y}} \times K_{\mathcal{X}} \subset \mathcal{Y} \times \mathcal{X}$.
2. When the set that the labels belongs to is a subset of a finite-dimensional real vector space, the probability density is continuous and the loss function is twice continuously differentiable.
3. For all $\mathbf{y}$ in $\mathcal{Y}$ (a finite set of labels) the probability density of $p(\mathbf{y},.)$ is continuous, and the loss function of $\ell_s(\mathbf{y},.)$ is twice continuously differentiable. In this notation, $g(\mathbf{y},.)$ denotes the function which associates to a vector $\mathbf{x}$ the value $g(\mathbf{y},\mathbf{x})$

Since the algorithm uses stochastic gradient descent, the optimization problem does not have to be convex. ℓ_s can be a convex loss function, such as square, logistic, or hinge.

If these assumptions are true, and the tuning parameters are selected properly, the algorithm will converge to the minimum values for the dictionary and model parameters. Based on the empirical results obtained, it appears the algorithm solves the problem of optimizing task-driven dictionary learning, and attains better performance on tasks as a result. It is mentioned that based on the tuning, different levels of error occur.

1.3.4 Relation to Course Material

Several topics covered in this term have been used in this paper and algorithm:

- LASSO
- Stochastic Gradient Descent
- Classification and regression
- Optimization problems

In a broad sense, this paper applies many of the techniques we learned about to an unsolved problem that then results in better performance on tasks we have also studied such as regression, classification, and data reconstruction.

This paper is mostly filled with ideas that I would have had little understanding of before taking this class, but are now within my reach (even if they take a bit to fully understand). Additionally there were things like the Cholesky decomposition that I didn't know about, but after some research were understandable. There is a whole larger context that this paper fits in that is still obscured to me, mainly because I haven't spent any serious time in the research world of this subject, but I think that is beyond the scope of whats possible in a term. I think this class gave me a base level understanding that would allow me to get some kind of foothold on any areas of the paper that aren't well understood, but aren't coming to mind.

2 Comparison of Algorithms

2.1 Ease of Implementation

While the three papers we read deal with the general topic of dictionary learning, the three groups approached the subject from very different angles. That being said, the algorithm outlined in the "Complete Dictionary learning via ℓ_4-norm Maximization over the Orthogonal Group," (CDL4) is by far the easiest to implement with a few SVD's being the primary cost of and implementation in the algorithm.

2.2 The Theory Within

CDL4 and "Trainlets: Dictionary Learning in High Dimensions" (TDLHD) both rely heavily on theory to inform the choices they make within their algorithm[3]. Both groups spend extensive time outlining the underlying theory and as a result they often find in their empirical data that the theory holds, so far. Both authors acknowledge that rigorous proofs global convergence are still forthcoming. Regardless, the empirical results of both are promising.

But, the goals of the two algorithms are very different. In CDL4 the authors are primarily interested in an efficient means of recovering the foundational truth dictionary and the sparse data set from which the output signal, which is the starting data in the algorithm, is constructed. So, the primary value in their algorithm lies in dictionary and signal recovery. Their algorithm tolerates some noise in the reconstruction of the dictionary which is then used to recover the original signal/data. So, in applications where a mild level of error and efficiency are highly prioritized, their algorithm will be very useful.

On the other hand, if accuracy is more important, then the algorithm proposed by TDLHD is the better choice. Accuracy is a primary focus of this algorithm. As well, they have created their algorithm for the specific task of image processing. Their primary goal is to increase the size and accuracy of images capable of being processed with a focus on subdividing the elements of the data set in a manner that improves their ability to ensure their intended accuracy during reconstruction as well as its efficient recovery. All the while, maintaining a steadily decreasing cost of computation. They show that through their algorithm "global dictionaries for entire images are feasible and trainable" without overtaxing training, memory, or computational load.

2.3 The Empirical

Both of these algorithms show admirable results towards their end goals. Both of these algorithms also test themselves on the MNIST data set though the TDLHD algorithm has observably better results. The CDL4 team did very well in the recovery of the original data set after about 20 iterations but with some random noise still visible, though this is within the goals set for their algorithm. However, the MSP algorithm from CDL4 has a more general construction and thus broader applications for images, data, or any signal that needs to be recovered from the output data, where the TDLHD is more geared towards image recovery and image compression with extreme accuracy.

"Task-Driven Dictionary Learning" (TDDL) takes a wholly different approach. Their work relies primarily on empirical data and frankly, is little more than a very slight modification of the already proven LASSO method. They introduce a new regularization parameter, λ_2, but then acknowledge that in most applications they have tried, save one, they set the parameter to zero. As well, they suggest that their method

does not require heuristic procedures to select parameters only after describing in detail the heuristic steps they took to create their suggested parameters for the few applications they tested. All that being said, they have adapted an algorithm that seems well suited to categorizing data with already known identifiers that are scale-able to as many identifiers as one may need. As a result, their algorithm could be useful in sifting through data for desired or intentionally tracked or already known variables.

2.4 Dealing with Complexity

In TDDL the authors attempt to address the inherent computational complexity in their algorithm though their success is questionable. They acknowledge that if not well initialized then their algorithm will yield poor results. However, they offer little more than some empirical observations of parameters they have used in several specific test examples with success. As well, they admit that supervised dictionaries have larger error than unsupervised dictionaries which they solve by using an unsupervised dictionary to build their initial supervised dictionary. And they never manage to deal with the error inherent in their algorithm. They fail to overcome the size limitations that often plague image processing with similar algorithms but do manage to reach an image size slightly larger than other algorithms in their class. As a result, this algorithm is only useful for small image processing or large-scale data labeling.

This stands in stark contrast to the TDLHD authors who very effectively deal with the inherent limits to algorithms of their type. Right at the outset they identify the two most prevalent shortcomings of the wavelet model and they then tactically eliminate those limitations for all practical purposes. The two limitations they identify are, one, the lack of separability of the wavelet transform and, two, the significant border effects inherent in wavelet transforms for small and medium size images. They solve the first by introducing a separable Kronecker product of two 1-D wavelets that are applied over two passes, the first over the rows of the image and the second over the columns of the result. This is also one of the methods they employ to solve the second issue inherent to wavelets. They further utilize zero padding to subdivide the image in order to introduce discontinuities which the algorithm utilizes to minimize the border effects generally inherent to the wavelet transform.

For a real-world example, I liken it to the act of zooming in on a drawing so that you can work in a higher scale so when you zoom back out you get a much crisper image at scale. By breaking the data down to the sparsest representations possible, they are able to reconstruct the image atom by atom with the resulting image remarkably close to the original. And these tweaks introduce a computational "simplicity" that makes the algorithm not only more accurate but also faster. As well, they manage to process images as large as 64x64 with the distinct possibility for larger image sizes with little tweaking.

The authors of CDL4 had very little computational complexity to contend with owing to the extremely streamlined method they illuminate. The only real computational hurdles exist by design or rather trade off. That hurdle being the noise/error in the data set. However, this occurs because the authors prioritize speed over accuracy. Regardless, they seem to find a consistent .35 % error over all the explored parameters (norm, data size, iterations, etc.) they explore in their data section, which is not outlandish. Meanwhile, the computational time is worlds faster than similar algorithms. This algorithm also seems to have almost limitless scalability due to its low initial cost.

3 Algorithm Implementation & Testing

We chose the ℓ_4-norm maximization algorithm. We chose this because it was easy to implement, fast to run, and John hasn't seen it yet. See code in A.2 Functions.

3.1 Results on Synthetic Data

This experiment shows tuning parameters vs performance. Tuning parameters are n, the number of feature dimensions, p, the number of samples, and θ, the sparsity level of the ground truth Bernoulli-Gaussian matrix $\boldsymbol{X_O}$. One of the tests of interest in the paper involved looking at the number of iterations necessary

to maximize either $\hat{f}(\boldsymbol{A}, \boldsymbol{Y}) = \|\boldsymbol{AY}\|_4^4$ or optimize $g(\boldsymbol{AD_O}) = \|\boldsymbol{AD_O}\|$, both of which are equivalent to recovering the ground truth dictionary.
We tested this using the following parameters:

- n = 50, p = 20000, $\theta = 0.3$
- n = 100, p = 40000, $\theta = 0.3$
- n = 400, p = 160000, $\theta = 0.3$

Each of the tests was performed with a maximum number of iterations of 60. As shown, as the input size increases, the algorithm takes longer to converge on the ground truth dictionary. This is not surprising, but the results did match the paper's which **was** a surprise. The original paper did not do the third test, but it was decided to try something deemed "too large" to see if it would find the ground truth dictionary under 60 iterations.

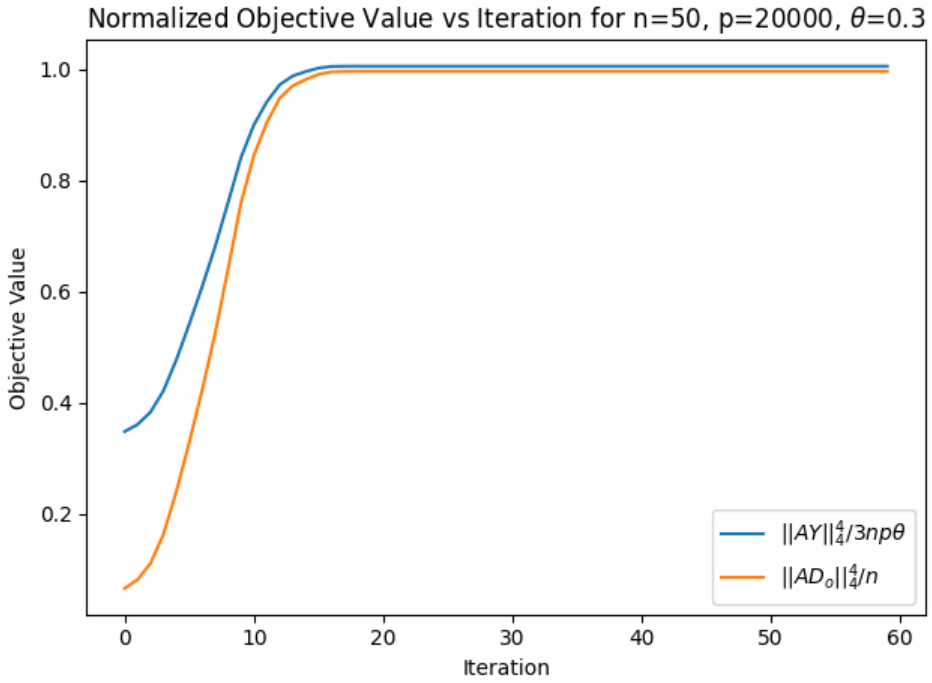


Figure 1: feature dimension =100, samples =20000, θ =0.3

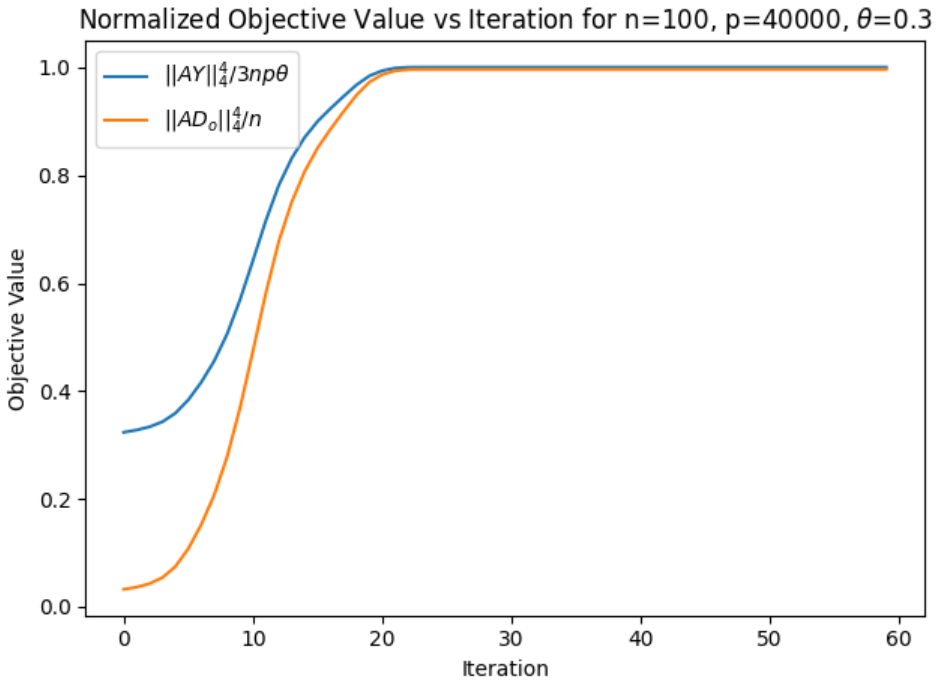


Figure 2: feature dimension =100, samples =40000, θ =0.3

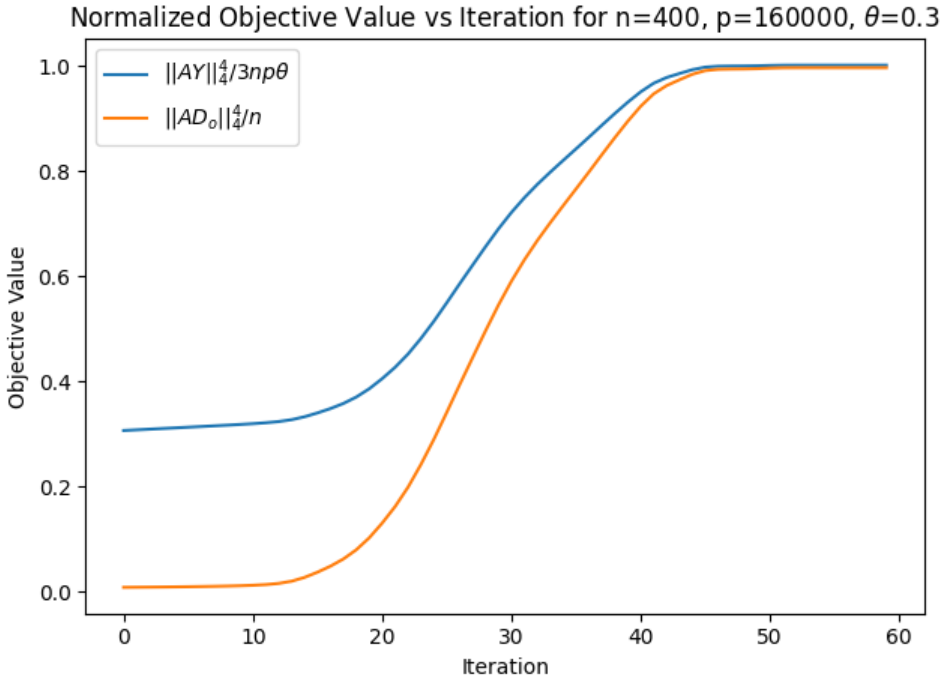


Figure 3: feature dimension =400, samples =160000, θ =0.3

Additionally, a heatmap was created showing the normalized error after 30 iterations with varying sample numbers and sparsity values. This heatmap shows a range of θ from 0 to 1 and a sample size from 500 to 1000, with a fixed feature dimension of 50. As shown, there is a clear line below which little error should be expected. There is some variability below the line that would resolve with more iterations.

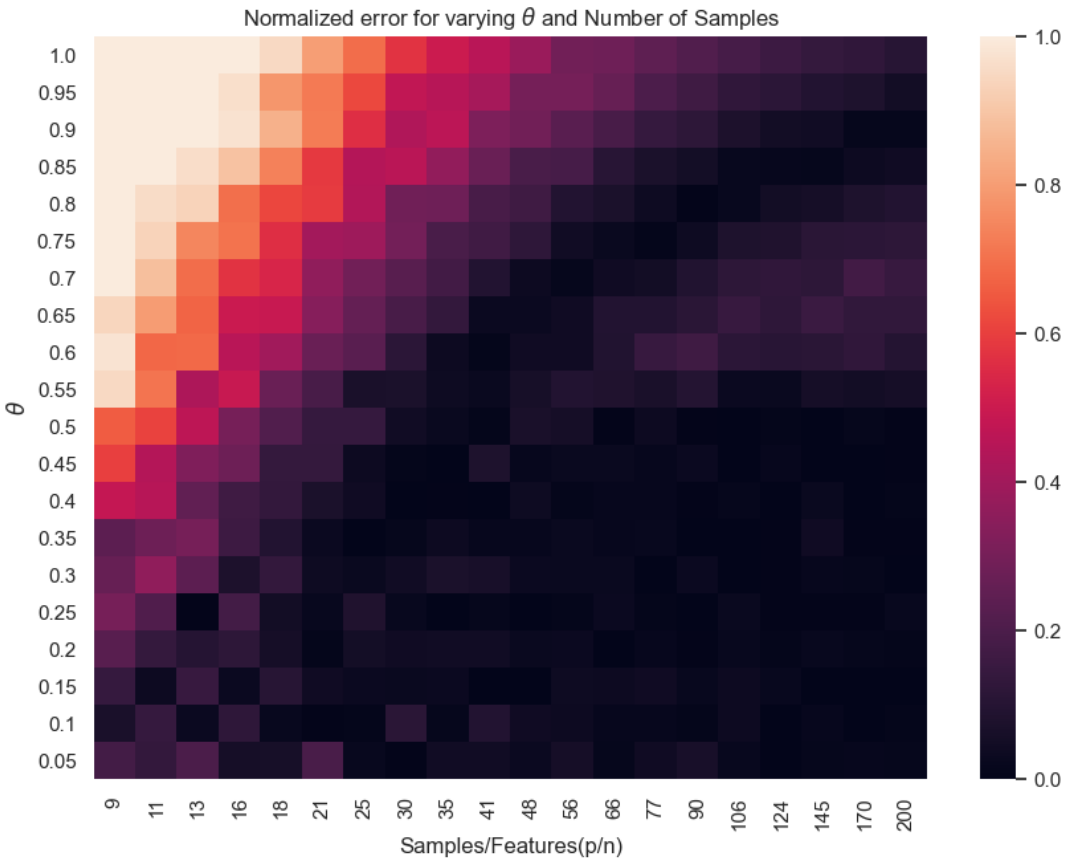


Figure 4: Heatmap showing error for various sample sizes and sparsity values

3.2 Results on Benchmark Data

As in the paper, a test was performed that involved comparing the top bases made from clean samples and noisy samples to compare the similarity of the bases.

Shown below, we found the top bases for the clean data set, and data sets with added noise from normal Gaussian distributions with standard deviations of 0.04173, and 0.3333. As the mean for the MNIST data set was 0.1394, this resulted in an SNR of 3.33 and 0.4181 respectively. The paper compares the top bases from the clean data set, and one with an SNR of 3.33. We decided to perform an additional comparison of one with a much worse SNR and surprisingly it still resulted in similar top bases.

For these tests, we used data sets of size 784x20000, and iterated the algorithm 50 times, based on our synthetic testing normalized objective value vs iterations plots.

Testing the error between a reconstructed image and the actual image using the learned dictionary, we

found an error of 9.2144e-24 for the clean learned dictionary, and 5.8289e-23 for the dictionary learned using an SNR of 3.33.

Figure 5: Top bases from clean MNIST set

Figure 6: Top bases from noisy MNIST set with SNR = 3.33

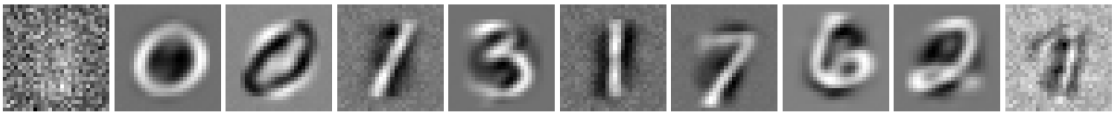
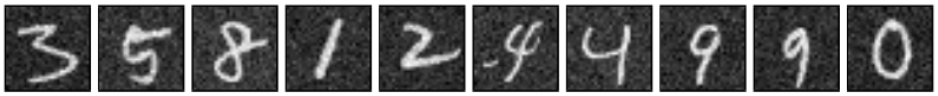


Figure 7: Top bases from noisy MNIST set with SNR = 0.4187

(a) Normalized MNIST to mean 0 and 1 std



(c) MNIST with noise, SNR = 3.333

(b) Top bases from MNIST

(d) Top bases from MNIST with noise

Figure 8: Example of results taken from [1] for comparison

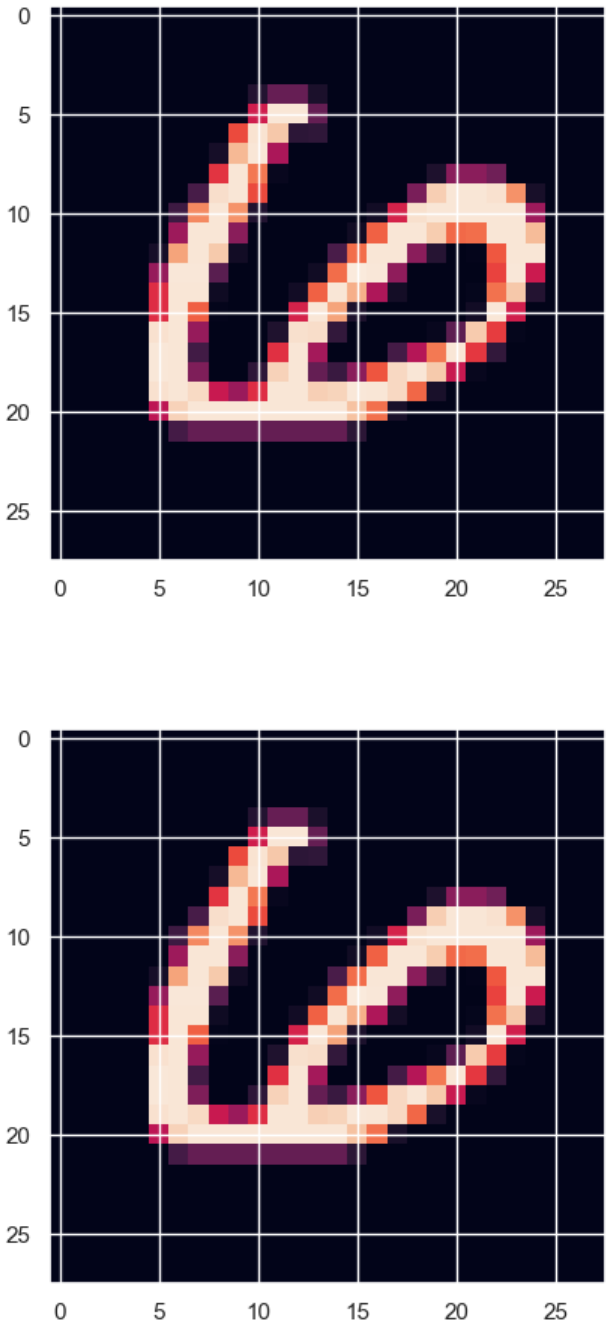


Figure 9: Comparison of original and reconstructed using dictionary learned with SNR = 3.33, original on top, reconstructed below

The run time for the this test was as follows:

- For the clean data set, with dimension 784x20000, the algorithm took 1.55 minutes to run 50 iterations.

- For the noisy data set, with dimension 784x20000, the algorithm took 1.63 minutes to run 50 iterations.

Our results are pretty close to what the paper achieved, at least visually. Any discrepancies can be chalked up to different sized datasets. They used the full 60,000 images from the MNIST dataset, whereas we used only 20,000. However, the quality of our reconstructed image and the minimal error shows that it was a fairly minimal difference, at least for this dataset.

Thanks for a great term and a challenging, interesting, and fun project to end it!

References

[1] Yuexiang Zhai, Hermish Mehtal, Zhengyuan Zhou, and Yi Ma. Understanding ℓ_4-based dictionary learning: Interpretation, stability, and robustness. 2020.

[2] Yuexiang Zhai, Zitong Yang, Zhenyu Liao, John Wright, and Yi Ma. Complete dictionary learning via l4-norm maximization over the orthogonal group. *Journal of Machine Learning Research*, 21(165):1–68, 2020.

[3] Wikipedia contributors. Orthogonal group — Wikipedia, the free encyclopedia, 2021. [Online; accessed 19-March-2021].

[4] Jeremias Sulam, Boaz Ophir, Michael Zibulevsky, and Michael Elad. Trainlets: Dictionary learning in high dimensions. *IEEE Transactions on Signal Processing*, 64(12):3180–3193, 2016.

[5] Julien Mairal, Francis Bach, and Jean Ponce. Task-driven dictionary learning. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):791–804, 2012.

[6] T. T. Cai and L. Wang. Orthogonal matching pursuit for sparse signal recovery with noise. *IEEE Transactions on Information Theory*, 57(7):4680–4688, 2011.

[7] Thanh Nguyen, Raymond Wong, and Chinmay Hegde. A provable approach for double-sparse coding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

A Appendix

A.1 Main Project File

```
#
    ================================================================================

# Authors:            Andrew, Mikhail, Anthony
# Last Updated:       Feb 19, 2021
#
# Notes:              Used for general tests and to show created atoms.
#                     Best bases extracted using code from [5]
#
    ================================================================================

import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse.linalg import svds, eigs
from tqdm import tqdm
import itertools

import random


'''
Used for dictionary initialization
Inputs:
  size of matrix
Returns:
```

```
  Orthogonal random matrix
'''
def radU(N): #call it radU for coolness
  X = np.random.randn(N,N)
  [Q,R] = np.linalg.qr(X)
  r = np.sign(np.diag(R))
  return Q * r.T

'''
l4_max is the MSP algorithm Described in paper
Inputs:
  Y is the observed data
  max_iter is the number of iterations
Returns:
  At is the approximated Learned dictionary
'''
def l4_max(Y, max_iter):
  M, N = Y.shape
  At = radU(M)
  for kk in tqdm(range(max_iter)):
    Aprev = At
    AtY = Aprev @ Y
    dA = (AtY**3) @ Y.T
    U, S, VT = np.linalg.svd(dA, compute_uv=True)
    At = U@VT
  return At
"""#MNISR Data Load"""

image_size = 28 # width and length
no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
data_path = "sample_data/"
train_data = np.loadtxt(data_path + "mnist_train_small.csv", delimiter=",")
test_data = np.loadtxt(data_path + "mnist_test.csv", delimiter=",")

fac = 0.99 / 255
train_imgs = np.asfarray(train_data[:, 1:]) * fac + 0.01
test_imgs = np.asfarray(test_data[:, 1:]) * fac + 0.01

train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])

img = train_imgs[0].reshape((28,28))
img = test_imgs[0].reshape((28,28))
print("Label: ",test_labels[0])
#plt.imshow(img, cmap="Greys")

X = np.array([[1,2,3],[4,5,6],[7,8,9]])

max_iter = 100
train = train_imgs
```

```
## Clean learning ##
D_patches = train_imgs

D_patches = D_patches.T

D_dictionary = l4_max(D_patches, max_iter) #actual learning being done
D_dictionary = D_dictionary.T
COLOR_SCALE = {
    'vmin': 0,
    'vmax': 255
}
D_sparse_encoding = D_dictionary.T @ D_patches
norms = np.abs(D_sparse_encoding)
norms = np.sum(norms, axis=1)
all_indices = list(range(len(norms)))
all_indices.sort(key=lambda row: norms[row], reverse=True)
sum_signs = np.sum(D_sparse_encoding, axis=1)
sum_signs = np.sign(sum_signs)
ROWS, COLS = 3, 4
fig, axs = plt.subplots(ROWS, COLS, figsize=(64, 48))
plt.subplots_adjust(left=None, right=None, bottom=None, top=None, wspace=0.05,
    hspace=0.05)
for index, ax in zip(all_indices, axs.flat):
    base = D_dictionary[:,index] * sum_signs[index]
    base = base - base.min()
    base = base / base.max() * 255
    base = np.reshape(base, (28,28))
    ax.imshow(base, **COLOR_SCALE, cmap='gray')
    ax.axis('off')
plt.show()

## Noisy learning ##
D_patches = train_imgs + np.random.normal(scale=.3333, size=train_imgs[0].
   shape)

D_patches = D_patches.T

D_dictionary = l4_max(D_patches, max_iter) #actual learning being done
D_dictionary = D_dictionary.T
COLOR_SCALE = {
    'vmin': 0,
    'vmax': 255
}
D_sparse_encoding = D_dictionary.T @ D_patches
norms = np.abs(D_sparse_encoding)
norms = np.sum(norms, axis=1)
all_indices = list(range(len(norms)))
all_indices.sort(key=lambda row: norms[row], reverse=True)
sum_signs = np.sum(D_sparse_encoding, axis=1)
sum_signs = np.sign(sum_signs)
```

```
ROWS, COLS = 3, 4
fig, axs = plt.subplots(ROWS, COLS, figsize=(64, 48))
plt.subplots_adjust(left=None, right=None, bottom=None, top=None, wspace=0.05,
    hspace=0.05)
for index, ax in zip(all_indices, axs.flat):
    base = D_dictionary[:,index] * sum_signs[index]
    base = base - base.min()
    base = base / base.max() * 255
    base = np.reshape(base, (28,28))
    ax.imshow(base, **COLOR_SCALE, cmap='gray')
    ax.axis('off')
plt.show()

## Reconstruct Image Using Noisy Dictionary ##
x = ss.lsqr(D_dictionary, train_imgs[0].T)
x1 = x[0]
x1 = x1.reshape(784,1)
testo = D_dictionary @ x1
testo = testo.reshape(28,28)
plt.imshow(testo)
plt.show()
plt.imshow(train_imgs[0].reshape(28,28))
plt.show()
```

A.2 Functions

```
#
    ================================================================================

# Authors:            Andrew, Mikhail, Anthony
# Last Updated:       Feb 19, 2021
#
# Notes:              MSP Algorithm from [1] As  interpreted by team.
#
    ================================================================================

import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse.linalg import svds, eigs
from tqdm import tqdm
import itertools

import random
'''
Used for dictionary initialization
Inputs:
  size of matrix
Returns:
  Orthogonal random matrix
'''
def radU(N): #call it radU for coolness
  X = np.random.randn(N,N)
  [Q,R] = np.linalg.qr(X)
  r = np.sign(np.diag(R))
  return Q * r.T

'''
l4_max is the MSP algorithm Described in paper
Inputs:
  Y is the observed data
  max_iter is the number of iterations
Returns:
  At is the approximated Learned dictionary
'''
def l4_max(Y, max_iter):
  M, N = Y.shape
  At = radU(M)
  for kk in tqdm(range(max_iter)):
    Aprev = At
    AtY = Aprev @ Y
    dA = (AtY**3) @ Y.T
    U, S, VT = np.linalg.svd(dA, compute_uv=True)
    At = U@VT
  return At
```

A.3 Generator

This was found after *projfuncs.py* was made. This was used for it's helper functions and as a reference.[1]

```
#
    ================================================================================

# Authors:          Hermesh, YX-S-Z
# Citation:         Understanding l4-based Dictionary Learning:
#                       Interpretation, Stability, and Robustness [5]
# Last Updated:     Feb 13, 2020
#
# Notes:            Used for helper functions.
#                   projfuncs.py was made before this was found
#
    ================================================================================

import numpy as np
from scipy.stats import ortho_group


def get_four_norm_maximization_iterates(size):
    """
    Runs the matching, stretching and projection (MSP) algorithm to
        iteratively
    maximize the (entry-wise) four norm of a SIZE by SIZE matrix, starting at
        a
    random orthogonal matrix, drawn from the Haar distribution. Returns an
    infinite generator which yields the current matrix at each iteration,
    beginning with random starting point.

    :param size: (int) The size of the matrix.
    :return: (generator) An infinite generator which yields all intermediate
        iterates of the algorithm.
    """
    current = get_random_orthogonal(size)
    yield current
    while True:
        delta_current = current * current * current
        left, _, right = np.linalg.svd(delta_current, compute_uv=True)
        projection = left @ right
        current = projection
        yield current


def get_dictionary_learning_iterates(observations):
    """
    Runs the matching, stretching and projection (MSP) algorithm to
        iteratively
    learn an orthogonal dictionary given a matrix of observations. Similarly
    starts at a random orthogonal matrix and iteratively improves the
    dictionary. Returns an infinite generator which yields the current
```

```
    dictionary at each iteration. Note that for practical purposes, this
       should
    be truncated.

    :param observations: (numpy.ndarray) A numpy array, where each column
        represents a new observation.
    :return: (generator) An infinite generator which yields all intermediate
        iterates of the algorithm.
    """
    current = get_random_orthogonal(len(observations))
    yield current
    while True:
        matched = current @ observations
        delta_current = (matched * matched * matched) @ observations.T
        left, _, right = np.linalg.svd(delta_current, compute_uv=True)
        projection = left @ right
        current = projection
        yield current


def random_dictionary_learning_instance(features, samples, theta):
    """
    :param features: (int) The number of features for each sample,
       equivalently
        the length of the signal.
    :param samples: (int) The number of samples or signals.
    :param theta: (float) The probability a particular entry in the decoded
        samples is non-zero. The smaller THETA is, the sparser the signals
        are in the optimal, intended, basis.
    :return: (tuple)
    """
    dictionary = get_random_orthogonal(features)
    samples = get_bernoulli_gaussian(theta, (features, samples))
    observations = dictionary @ samples
    return observations, dictionary, samples


def sum_of_fourth_powers(matrix):
    """
    :param matrix: (numpy.ndarray) A numpy array.
    :return: The fourth power of the four-norm of the matrix. In other words,
        the sum of the fourth power of all of its entries.
    """
    squared_entries = matrix * matrix
    return np.sum(squared_entries * squared_entries)


def get_random_orthogonal(size):
    """
    :param size: (int) The dimension of the matrix.
    :return: (numpy.ndarray) Returns a random orthogonal matrix from O(SIZE),
```

```
        the orthogonal group of dimension SIZE, drawn from the Haar
        distribution. The matrix has size (SIZE, SIZE).
    """
    return ortho_group.rvs(size)


def get_bernoulli_gaussian(theta, size):
    """
    :param theta: (float) The probability a particular entry is non-zero: must
        be between 0 and 1 inclusive. The smaller THETA is, the more sparse
            the
        output will be in expectation.
    :param size: (int or tuple) The shape of the output.
    :return: (numpy.ndarray) A random numpy array where each entry is from
        independently and identically distributed according  to a bernoulli-
        gaussian
    """
    bernoulli = np.random.binomial(1, theta, size)
    gaussian = np.random.standard_normal(size)
    result = bernoulli * gaussian
    return result
```

A.4 Synthetic Test

```
#
    ================================================================================

# Authors:              Andrew, Mikhail, Anthony
# Last Updated:         Feb 19, 2021
#
# Notes:                Used for synthetic tests and plots.
#                       Used to test changes of features and number of samples.
#
    ================================================================================

import generator as g
import numpy as np
import matplotlib

import matplotlib.pyplot as plt
import os
from projfuncs import l4_max, radU
import seaborn as sns
from tqdm import tqdm


features = 50
s = 20000
t = 0.3
# samples = np.linspace(500,10000,20, dtype=int)
# theta = np.linspace(0,1,20)
# htmp1 = np.zeros((20,20))
# htmp2 = htmp1.copy()
err1 = []
err2 = []
max_iter = 60
# for i,t in enumerate(theta):
#     for j,s in enumerate(samples):

Y, D, X  = g.random_dictionary_learning_instance(features, s, t)

# my_D = l4_max(Y, max_iter)
#l4_max with other error
M, N = Y.shape
At = radU(M)
for kk in tqdm(range(max_iter)):
    Aprev = At
    AtY = Aprev @ Y
    dA = (AtY ** 3) @ Y.T
    U, S, VT = np.linalg.svd(dA, compute_uv=True)
    At = U @ VT
    err1.append(g.sum_of_fourth_powers(At @ Y) / (3 * features * s * t))
    err2.append(g.sum_of_fourth_powers(At @ D)/features)
```

```
# my_D = my_D.T
# print(abs(1-g.sum_of_fourth_powers(my_D@D)/features))

# print(g.sum_of_fourth_powers(my_D @ Y) / (3 * features * s * t))
# htmp1[i,j] = abs(1-g.sum_of_fourth_powers(my_D@D)/features)
# htmp2[i,j] = g.sum_of_fourth_powers(my_D@Y)/(3*features*s*t)

# sparseX = my_D.T @ Y
# print(np.linalg.norm(Y-my_D @ (my_D.T @Y)))
# print(np.linalg.norm(Y-D@X))
# plt.imshow(htmp1)
# plt.show()
# plt.imshow(htmp2)
# plt.show()

plt.plot(err1, label=r'$||AY||_4^4/3np_\theta$')
plt.plot(err2, label=r'$_||AD_o||_4^4/n$')
plt.xlabel('Iteration')
plt.ylabel('Objective_Value')
plt.title('Normalized_Objective_Value_vs_Iteration_for_n={},_p={},_$\\theta$
    ={}'.format(features,s,t))
plt.legend()
plt.show()
```