LISTA: Concerns and Extensions

Zachary Levine

February 5, 2023

Contents

1 Introduction

1.1 Main

In *Learning Fast Approximations of Sparse Coding*, Karol Gregor and Yann LeCun propose two algorithms that produce approximates of sparse codes based on unfolding the well-known Iterative Shrinkage Thresholding Algorithm (ISTA) and Coordinate Descent algorithms, respectively, into deep nonlinear neural networks [2]. The problem setting is that of sparse coding, a similar and interrelated discipline to compressed sensing. We will use the terms ISTA and FISTA interchangeably throughout the result, though our technical implementation in the result section is of FISTA [1].

1.2 Relation to Compressed Sensing

Sparse coding is the problem of taking an input vector and reconstructing it by a linear combination of overcomplete basis vectors with sparse coefficients. Given an input vector y, (the measurement vector) we can see that this problem is identical to finding a matrix A and a sparse vector x such that $y = Ax$, or minimizing the difference between the two with a penalty on the norm of x. The difference between sparse coding and compressed sensing is subtle but important. In compressed sensing, the dictionary matrix A has much fewer rows than columns, so that it is underdetermined. In sparse coding, we do not have this constraint, indeed we can have more rows than columns in our matrix A.

2 The good

The numerical results presented in this paper present very strong evidence for why the two algorithms presented are useful. The presented algorithms converge in less time and iterations than the standard ISTA and CoD algorithms, and the authors make the case for this very well. My comments on the article are more on the theoretical grounding of the algorithms than numerical results.

3 Theoretical concerns

My overall complaint with this paper is a theoretical one. One of the major arguments presented in here is that of a straw man argument, which occurs when one refutes an argument, but does so by replacing it with a false or weaker one. The authors present the strength of LISTA and learned coordinate descent by comparing them to two simple encoder architectures which are based on a single coordinate-wise nonlinearity. These encoders are very weak, so of course LISTA and LCoD are better. The authors present baseline encoders with two problems, they argue: the difficulty of obtaining code values near zero, and lack of competition between different subsets of code components whose corresponding basis functions reconstruct the input equally well.

The first problem is a result of using $\tanh(x)$ as the activation function in the baseline encoder, however using this activation function was the author's own choice. Furthermore, these counter example encoders, as presented, do not even seem suitable for the task of sparse coding. If the encoders presented as comparison examples don't produce values close to zero very easily, why do the authors even suggest using them for sparsity problems where we know we want to output zero for many values in the code?

The second theoretical strength of the two encoders presented in the paper (as opposed to their dominance in numerical results) as argued by the authors, is that they implement "an approximate explaining away (or competition) between code components, so that if two sets of basis vectors could reconstruct the input equally well, one set will be picked by the algorithm, while the other one will be suppressed" (2). And while this point is likely true, it is merely stated at first without any supporting argument. The authors move on to discussing the architecture of the neural network without discussing this point more, but this is important. In section 3.2, the authors elaborate further on this important point as follows.

First, they state that: "In overcomplete situations, it is necessary for the inference algorithm to allow the code components to compete to explain the input" (4). My complaint with this statement is minor and does not affect the argument the authors are presenting here, but it seems that in all cases we would want code components to compete with each-other, not in only the overdetermined case. The authors could have worded this statement better. My main complaint here that the authors then say that "The baseline encoders have no such capability, which is the reason behind their second shortcoming as mentioned above." But the baseline encoders are the authors own proposal. If the authors had compared at baseline to encoders which did not have this problem, what would be the theoretical strength of LISTA?

The author's argument for why the baseline encoders do not allow for competition between code components goes as follows. Assume we have an input that contains an edge very close to a row of A. If two rows of A contain very similar filters, the baseline encoder will activate both code components equally. However, we really want here to activate one filter and turn off the other one, "explaining it away" by the first. The authors argue that the encoder will do this because activations of the two code components are independent given the input, and lack interaction terms. My question is where are the interaction terms in LISTA and LCoD? Aren't activations still independent between different code segments of the same iteration? Nowhere in the paper is this explicitly pointed out, and I would have found that helpful in my analysis.

This is not to say that LISTA is useless, indeed the numerical strength is presented and argued well in the paper. But the theoretical grounding of the encoders seems slightly shaky, for the reasons presented above.

4 Practical concerns

To do LISTA (or LCoD), according to the paper, one should first learn a dictionary from image patches through proximal gradient descent. This important step is swept under the rug in the paper, even though the results and success of LISTA and LCoD greatly depend on it. If the dictionary is badly learned, then all the training data goes out the window, because FISTA won't work to generate the ideal codes.

After training the dictionary W_d, one must train an encoder to predict the true FISTA samples from LISTA samples. This highlights an important point. LISTA and LCoD are not actually learning to encode and find sparsity in data, but simply learn mimic other algorithms which do the job already. Because they depend on ISTA and CoD, any drawbacks that occur in the traditional algorithms will by definition happen in the LISTA presented in the paper. For instance, as presented by Beck and Teboulle (2009), a drawback of traditional LISTA and FISTA is that the Lipschitz constant $L(f)$ is not always known or computable, for instance it depends on the maximum eigenvalue of $A^T A$, which for large problems is not always easily computable. If this quantity, is improperly computed or approximated LISTA will suffer just as heavily, if not more, than ISTA. What is the point of learning a dictionary matrix in LISTA if the matrix we are training against, the input to FISTA, is fixed?

Furthermore, what is benefit of learned methods if one still has to train a dictionary, and then run LISTA (or FISTA) thousands of times to train the encoder? Why shouldn't one just use FISTA or CoD to begin with, without bothering to train a neural network?

Lastly, the improvements in LISTA or LCoD over ISTA and CoD are presented in terms of fewer iterations required to reach the same prediction error in codes. But this is done without accounting for the initial steps of proximal gradient descent required to train W_d. It is likely that this would affect the numerical dominance of the learned methods over their traditional counterparts. For both traditional and learned ISTA, one needs to train W_d, but perhaps one could take fewer proximal gradient steps and spend less time training W_d if one was using ISTA instead of its learned counterpart? What is the sensitivity of LISTA to the amount of computation and time spent training W_d as compared to ISTA? This sensitivity analysis was missing in the paper.

5 Expansion

5.1 Introduction

A more interesting and useful learned ISTA algorithm would be able to learn ISTA implicitly from the data without having to run ISTA along side training the encoder to give as training data. This would also remove the dependency on the learned parameter W_d. The deep learning architecture that comes to mind for this task is an autoencoder. These models learn an identity map from the data, mapping inputs to themselves without needing labelled response or in this case "ideal codes," for each sample to learn against. By setting the architecture properly to represent ISTA, we could theoretically learn sparsity from the data without needing to give the algorithm already ran ISTA samples.

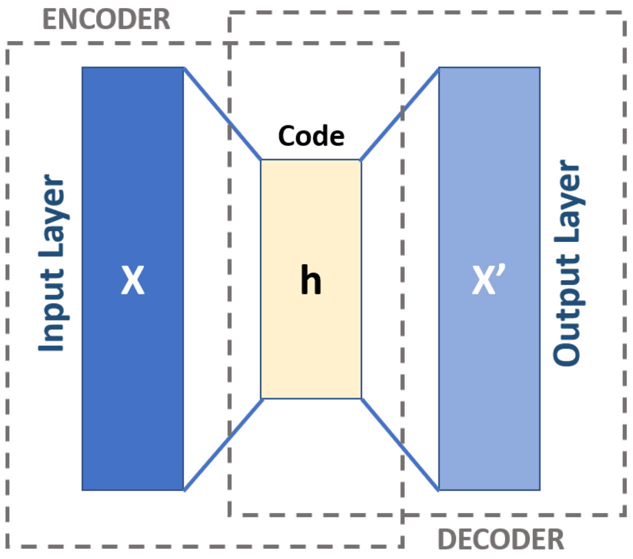


Figure 1: The basic structure of an autoencoder [3]

We would like to have an autoencoder that unfolds LISTA in its forward pass, and then reconstructs the input. In the next section, we will present out experiment designing this autoencoder, comparing our results against the natural alternatives: standard FISTA and LISTA implemented in Python.

5.2 Data

To match the original paper, we chose 321×481 images from the The Berkeley Segmentation Dataset (link), set them to grayscale, and cut each image into patches of 10×10 pixels. We stored both the original images and their patches in two different directories. We flattened images into vectors of dimension $(100, 1)$, and aggregated n image patches into batches of dimension $(100, n)$, so each column in a batch of images corresponded to a different image patch. We chose Python and used the deep learning library Pytorch for all our computation.

Figure 2: Four sample images from the Berkley Segmentation Dataset

5.3 Methods

We chose FISTA as the baseline example because it converges faster than ISTA. Our standard implementation for FISTA is presented below. We begin with an implementation of the shrinkage function that is a hallmark of this approach.

```
def shrink(V, theta):
     h = np.sign(V) * np.maximum(np.abs(V)-theta, 0)
     return h
```

Then, our FISTA implementation, in pseudocode is

```
def fast_ista(X, W_d, alpha=0.01, max_iters = 3000):
  Z starts off as a random normal matrix
  for each iteration:
    Take a step for Z
    shrink(Z, theta/L)
 return Z
```

Where we approximate the Lipschitz constant L by $L(f) \approx ||W_d||_2^2$, which is a bound on $\rho(A^T A)$, the maximum eigenvalue of $W_d^T W_d$. We consider W_d as a real matrix here, so that its adjoint $W_d^* = W_d^T$.

Proof:

Let $\rho(W_d * W_d) = \lambda_1 \geq \lambda_n \geq 0$ be the eigenvalues of $W_d^T W_d$. Then:

$$\begin{aligned} ||W_d||_2^2 &= \max_{||x||_2=1} ||W_d x||_2^2 \\ &= \max_{||x||_2=1} \langle W_d x, W_d x \rangle \end{aligned}$$

By the definition of the adjoint, $\langle W_d x, W_d x \rangle = \langle W_d^T W_d x, x \rangle$, so

$$\begin{aligned} ||W_d||_2^2 = \max_{||x||_2=1} \langle W_d^T W_d x, x \rangle \\ \leq \lambda_1 \end{aligned}$$

Where the last inequality is a direct result of applying the Rayleigh-Ritz theorem.

Both FISTA and LISTA depend on learning a dictionary. For conciseness, the pseudocode implementation for the training of W_d is provided below, aside from the proximal gradient descent. We again use our estimate developed in the previous section for L.

```
def proximal_gradient_descent(X, Z, A, alpha):
  A2 = np.matmul(A.T, A)
  L = (np.linalg.norm(A, 2) ** 2)
  temp1 = np.matmul((np.eye(A.shape[1]) - (1/L)*A2),Z),
  Z = shrink((1/L) * np.matmul(A.T, X) + temp1, theta = alpha/L)
  return Z

def learn_Wd(num_epochs, alpha, lr, beta):
     set num iters, dimensions of dictionary
     for each epoch
           a set of 10 random patches
           run proximal gradient descent to find Z
           run gradient descent on the dictionary
```

We found that training on a random sample of 10 patches from our set yielded superior results as opposed to all patches from one image for which the training set was much larger. This is likely because a random sample of patches may better represent the distribution of all patches than patches from a single image.

We trained our dictionary and then ran FISTA for several images by cutting them into patches, finding the sparse codes for each patch, multiplying the sparse codes by W_d and and then reconstructing the result from its constituent patches.

As a second comparator, we implemented the LISTA model as described by Yan and LeCun in Pytorch with ten unfolded iterations. We trained for 10 epochs on random patches of 10 images with an Adam Optimizer and a learning rate of 0.001. Results are included in the results section.

5.3.1 First Attempts

We tried to train an autoencoder based on the identical architecture presented by Gregor and LeCun, with a simple multiplication by the learned W_d as the final layer. We set the loss to be the standard MSE loss between the original image and the reconstruction, instead of against the FISTA sample. This approach did not work at all. While the model could learn codes for patches and reach a minimum MSE of 0.001, the reconstructed images looked nothing like the original images. In fact, the reconstructed images all were composed of horizontal lines with zero likeness to the original image. Our results suggest that more complex and perhaps domain-specific architecture is required to learn ISTA without training data.

In Learned Convolutional Sparse Coding, Hillel Sreter and Raja Giryes consider this problem. They propose a novel convolutional recurrent sparse autoencoder, a convolutional extension of LISTA with a linear convolutional decoder. We used their method of recasting ISTA iterations as convolutions by learned matrices, and took inspiration from them for the exact number of channels of the convolutions [4]. Our architecture is provided below.

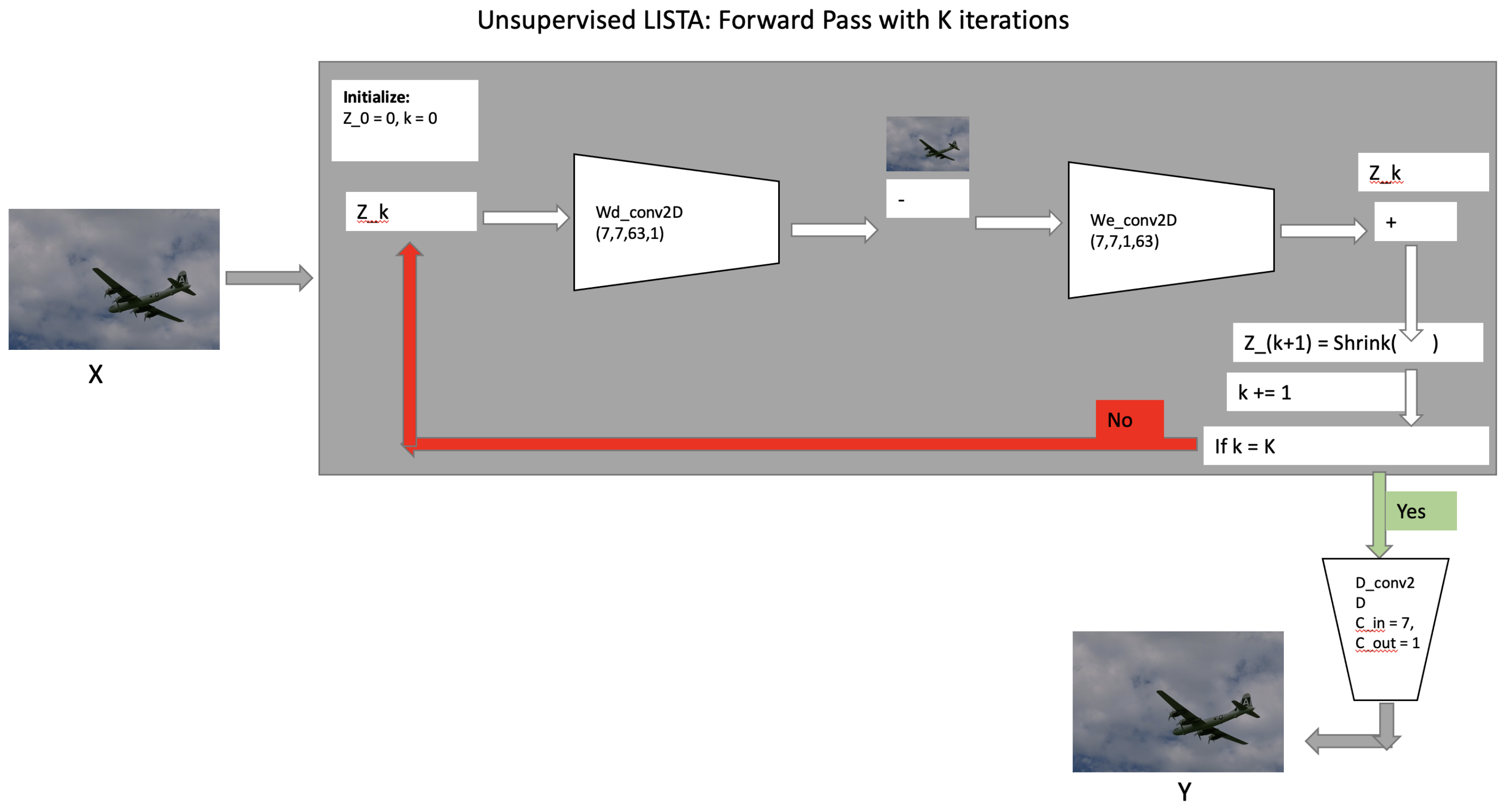


Figure 3: Architecture of the unsupervised convolutional autoencoder model.

Z_0, the first iteration of the forward pass is set to a zero matrix. Then, for the kth iteration, we pass Z_k through a 2D convolution with the learned weight matrix W_d, that increases the channels of the image (in this case 7). This result is subtracted from the original image and passed through another convolution, W_e that operates as well on seven channels. After that, the result of this convolution is added to Z_k, and is sent through the shrinkage function. If $k = K$, in other words if we have done all our iterations, then we convolve this result by a third 2D convolution, with a learned weight matrix. This final convolution sets the channels back to the original number, 1 (as we are dealing with greyscale images), essentially performing the multiplication by W_d that happens at the end of FISTA/ISTA to recover the original image.

In summary, in the forward pass of the network, we replace standard multiplication in ISTA by convolution by two learned dictionary matrices which start out as transposes of oneanother, and iterate through several unfolded iterations of convolutional ISTA. To get back the original image we convolve again by a third learned weight matrix.

We trained this network on the same Berkley Segmentation dataset, for each epoch grabbing a random image, using a simple MSE loss and an Adam optimizer with learning rate of 0.001, for 10 epochs with 10 iterations of unfolded convolutional ISTA performed in each forward pass. Training this way takes a total of 1 minute and 20 seconds. This network does train, but we wondered how well, and how our results compared to standard LISTA.

5.4 Results

We can see that the filters from two dictionaries seem to be similar in structure.

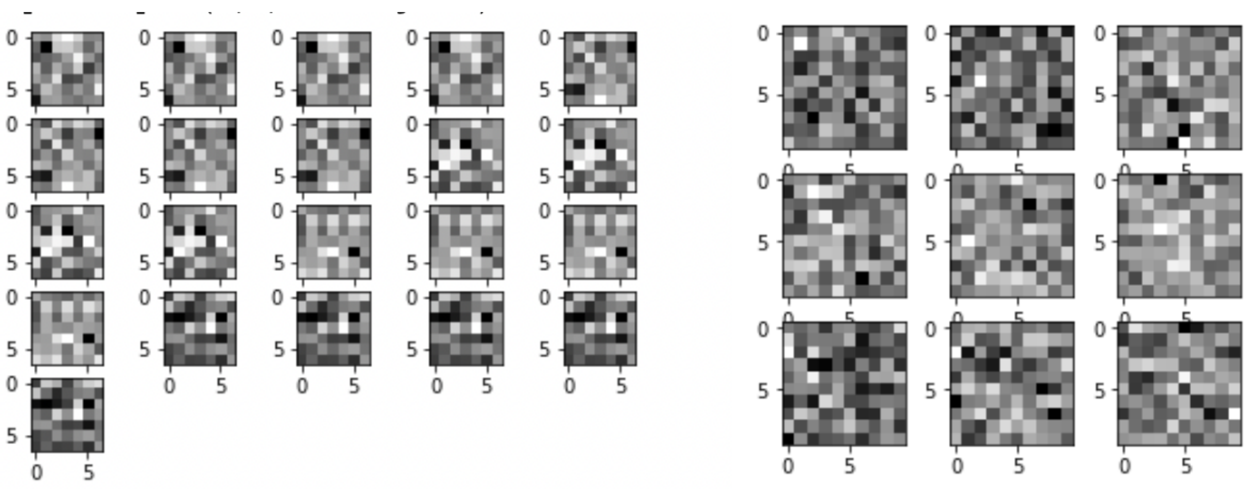
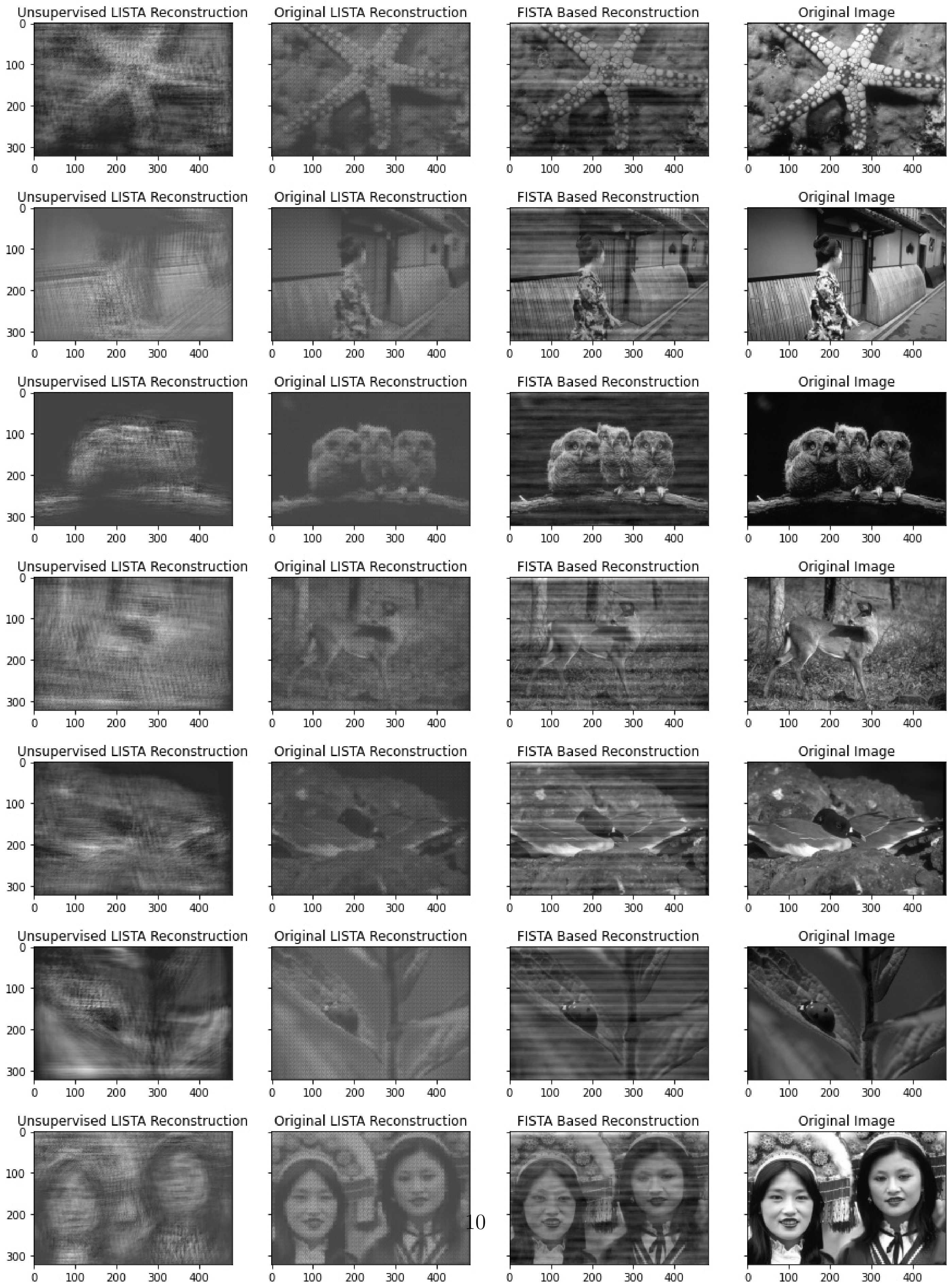


Figure 4: The first 25 filters from the learned dictionary used as the weight matrix in convolution W_d (left). The first 9 filters from the learned dictionary W_d from the Berkley Segmentation Dataset based on 10 epochs, sparsity parameter $\alpha = 0.00001$, and a learning rate of 0.01 (right).

We compare the autoencoder to LISTA in three metrics: visual reconstruction similarity, reconstruction error per epoch, and sparsity of solutions obtained.


Unsupervised LISTA Reconstruction
Original LISTA Reconstruction
FISTA Based Reconstruction
Original Image
0
100
200
300
400

First, as seen in the above results, we can see that our reconstruction does recover some of the original image, but the original LISTA and FISTA reconstructions are closer to the original image than given epochs, learning rate, and iteration numbers. Perhaps more training or different hypte parameters is required. Still, the fact that the model was able to learn anything at all is interesting, because there was was no supervisory signal telling the network to do LISTA.

Second, we wanted to understand terms of reconstruction loss per epoch. For each epoch, we checked the MSE between the reconstruction of a randomly-selected image that was held-out from the training set, and the true image.

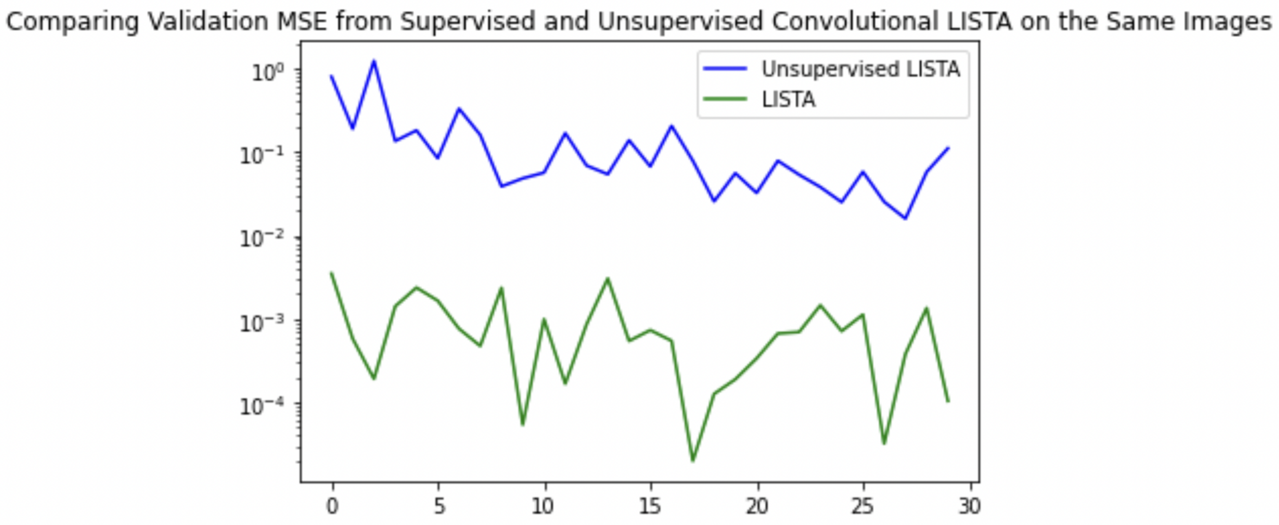


Figure 6: Comparing validation loss between standard LISTA and convolutional unsupervised LISTA on the same images with identical training parameters over 30 epochs. The y axis of the figure is shown on a log scale.

From this illustration we can see that overall, all things being equal, original LISTA produces much more accurate codes than our implementation. Lastly, we also were concerned with the average sparsity of solutions per epoch, defining sparsity as the proportion of values that were zero in the solution of the model. On sets of identical images, we compared the sparsity of solutions from the autoencoder to LISTA.

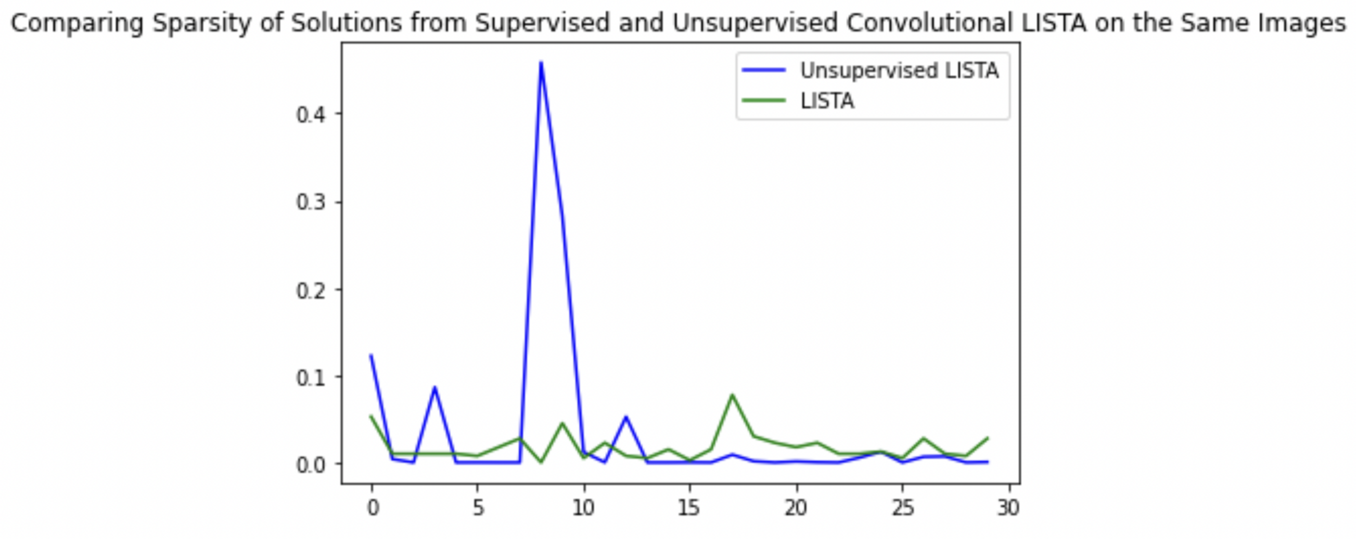


Figure 7: Comparing sparsity between standard LISTA and convolutional unsupervised LISTA on the same images with identical training parameters over 30 epochs.

From this illustration we can see that overall, all things being equal, there is similarity between the sparsity of solutions from supervised and unsupervised LISTA. However, where sparsity of the estimates is nonzero, unsupervised LISTA often outperforms supervised LISTA on the same images. This concludes our extension of the work.

6 Conclusion

Here, we considered practical and theoretical remarks on the paper introducing LISTA by Gregor and LeCun. It was argued that while there is value in LISTA in terms of its numerical dominance over ISTA and other non learned methods, the main theoretical grounding presented by the authors is shaky. Inspired by more practical concerns, we considered an extension of LISTA that is unsupervised based on convolutional autoencoders, and saw that it underperformed compared to standard LISTA and FISTA on the same number of iterations in terms both visual reconstruction similarity, accuracy, but matched LISTA on sparsity. Further investigation is required to confirm our results and what they mean.

References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183?202, Jan 2009.

[2] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML?10, page 399?406, Madison, WI, USA, Jun 2010. Omnipress.

[3] Michael Massi. Papers With Code- Autoencoder. 2021.

[4] Hillel Sreter and Raja Giryes. Learned convolutional sparse coding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 2191?2195, Apr 2018. arXiv:1711.00328 [eess].