Sparse Coding

Applied to Digit Recognition

Clement Gehring (260414522) Simon Lemay (260430046)

April 26, 2012

1 Introduction

Through the works of researchers such as Hubel and Wiesel [1], we now have a good idea of the mechanisms at play in the early vision process. Sparse coding is an interesting technique for computer vision as it seems to emulate these mechanisms. In this section, we will explain what sparse coding is and why sparseness is an important property. We will also describe the real world problem we set out to solve using this technique.

1.1 Sparse Coding

Sparse coding as a vision technique is an idea first introduced by Olshausen and Field [2, 3]. As with multiple approaches (PCA for example, and linear models in general), sparse coding's representation of an image uses what we could call a basis: a set of images that capture some features/characteristics/properties of the original image(s). In sparse coding, linear combinations of these bases are used to represent an image:

$$x = \sum_{i=1}^{n} s_i b_i$$

Here, x is the represented image, the b_i is the i-th basis, n is the number of bases and the s_i's are their corresponding coefficients in the linear combination. The particularity with sparse coding is that it allows the basis to be *overcomplete* and requires the coefficients to be *sparse* (most of the s_i's should be 0). The motivations for this will be discussed in the next subsection. Also note that this definition doesn't directly lead to an algorithm for computing the bases of a given set of images. Thus, (efficiently) finding bases that can handle sparse representations is often difficult and is the main problem we were confronted to during our project. An example of bases obtained with sparse coding is given in figure 1.

1.2 Why Sparseness?

Let's first consider what it would mean to have a non-sparse representation. Images would be combinations of a lot of basis images, and there could be complex statistical dependencies and redundancy between those. Analysing those representations to extract meaningful information would be hard as we would need to detect and make sense out of all these dependencies. Intuitively, enforcing that the representation be sparse (while still relatively exact) means that we prefer bases

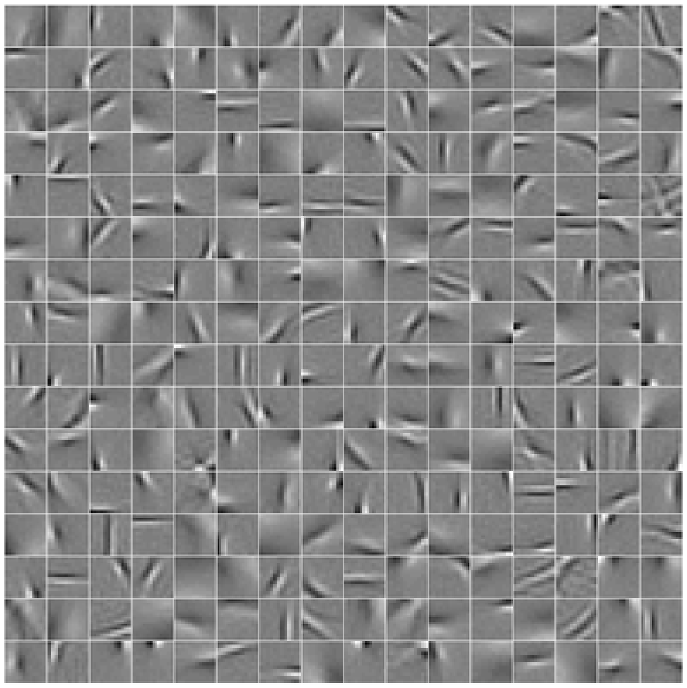

Figure 1: An example of bases computed with sparse coding. The bases was obtained from training on natural images. Note the resemblance of each little image to the typical description of receptive fields in the visual cortex's simple cells. The image is taken from [5], and all details on how it was computed can be found there.

that are statistically independent from each other, since we express the most information with the least amount of components. Thus, sparseness leads to a reduction in redundancy and statistical dependencies [2].

While making the representations simpler in a statistical way is nice, sparseness is also interesting because it could well be that natural images typically have a sparse structure. Intuitively, we can think of most natural images as easily described in terms of a few primitives such as lines and shapes. A sparse representation could then pick up only those few primitives that best describe a given image. This intuition seems to be confirmed by some experiments indicating that natural images have a sparse structure [4].

In fact, some things hint that sparse coding might be (close to) the representation of images in the early visual process. The main argument for this claim being that basis computed with sparse coding are similar to the receptive fields of simple cells in the visual cortex [2, 3], as can be seen in figure 1. Other reasons sparseness may be a property used in the brain is increased capacity in associative memory [6], easier formation of short neurons [7] and metabolic efficiency [8].

Although this is not really related to sparseness, another interesting property of sparse coding is that it allows an *overcomplete* set of bases. This gives more flexibility in choosing bases that match the structure of the input images and more "tolerance" regarding noise/degradations [2]. Furthermore, in some cases, overcomplete bases allow representations that better model the statistical density of the data [9].

1.3 Task

The task we set out to solve is handwritten character recognition. We have limited ourselves to the MNIST dataset[17] which contains 70000 labelled handwritten digits from zero to nine. The digits come from approximately 250 different writers. This dataset is convenient since quite a few different methods have been tried on it which will allow us to compare our results to the more advanced methods. We split the database into 60000 training images and our testing set thus contained 10000 images.

2 Algorithm

The first step to character recognition using sparse coding is to obtain a bases: a set of images with which we can sparsely represent every image in our training set (and hopefully in our testing set too). Once we have this, we're left with a simple classification problem: using the sparse representations of our labelled character images, find the label (the character) associated with the sparse code of an input image. This section will cover the algorithms we used for both problems, with an emphasis on obtaining a the sparse bases (as it most relates to sparse coding).

As notation, for this section, B is a matrix where each column $\vec{b}_j$ represents a basis, X is a matrix where the j-th column $\vec{x}_j$ is the j-th input image and S is a matrix of coefficients, where the column $\vec{s}_j$ represents the coefficients for each basis in the sparse code for $\vec{x}_j$.

2.1 Computing the Bases

The task of finding B can be described as finding the best possible set of images that can handle a sparse representation of the input images X. This is best described mathematically by the following optimization problem:

$$\min_{B,S} \ ||X - BS||_2^2 + \lambda \sum_j \phi(\vec{s}_j), \text{ subject to } ||\vec{b}_j||_2^2 \leq c, \ \forall j \in [1..n]$$

Here, n is the number of bases, ϕ is a sparsity-inducing function (more on that in a moment), c is a constant and λ is a regularization parameter which can be modified to give more or less importance to the sparsity of the coefficients. Minimizing the first term, $||X - BS||_2^2$, ensures that the sparse representations BS are as close as possible to the original images X. The second term enforces sparse coefficients through the use of ϕ, a function which is minimum when its input is sparse. Different choices of ϕ are possible, the most popular being the ℓ-1 norm, the ℓ-2 norm and the so-called ℓ-0 norm (the number of non-zero entries in a vector). The constraint might seem a bit arbitrary, but is necessary to avoid growing B without bounds. Without it, we could always find a larger B (with S tending to 0) minimizing the above [10], which is not something we want. An alternative would have been to impose some variation of the coefficients for each image in the bases, as in [2].

This is a hard problem (if only because there are a lot of variables), and it turns out it can be made a lot simpler by fixing B or S [2, 10]. When fixing S and optimizing B only, we have a least squares problem and when doing the opposite (fixing B, optimizing S), we have a "ϕ-regularized" least squares problem. Both problems (especially when ϕ is the ℓ-1 norm) have been extensively studied in the literature and we have reasonably efficient algorithms to solve them (or at least find approximations).

The algorithm to find B will thus alternate between learning B and S while holding the other fixed. As we iterate this process, our approximations of B and S will get better and better. The following subsections will detail the approach used to solve each sub-problem.

Learning B: Lagrange Dual

If we fix S, we can drop the second term of the optimization problem above to obtain the reduced minimization

$$\min_B \; ||X - BS||_2^2, \text{ subject to } ||\vec{b}_j||_2^2 \leq c, \; \forall j \in [1..n]$$

which we rewrite as

$$\min_B \; \text{trace}\left((X - BS)^{\mathrm{T}}(X - BS)\right), \text{ subject to } ||\vec{b}_j||_2^2 \leq c, \; \forall j \in [1..n]$$

We'll solve this using a Lagrange dual, as proposed in [10]. We first write the constraints as Lagrange multipliers to get

$$\min_B \; \text{trace}\left((X - BS)^{\mathrm{T}}(X - BS)\right) + \sum_j \lambda_j(||\vec{b}_j||_2^2 - c)$$

Each $\lambda_j \geq 0$ is a dual variable (not to be confused with the regularization parameter of the original problem). We then analytically minimize over B (and thus over the $\vec{b}_j$'s too) to get the Lagrange dual problem

$$\max_{\vec{\lambda}} \; \text{trace}\left(X^{\mathrm{T}}X\right) - \text{trace}\left(XS^{\mathrm{T}}(SS^{\mathrm{T}} + \Lambda)^{-1}(XS^{\mathrm{T}})^{\mathrm{T}}\right) - c\sum_j \lambda_j = \max_{\vec{\lambda}} \; \mathcal{D}(\vec{\lambda})$$

where $\Lambda = \text{diag}(\vec{\lambda})$ and subject to the constraint that each element of $\vec{\lambda}$ is ≥ 0. This can then be solved using any off-the-shelf function optimizer. Once this is done, B can be retrieved using the following formula:

$$B^{\mathrm{T}} = (SS^{\mathrm{T}} + \Lambda)^{-1}(XS^{\mathrm{T}})^{\mathrm{T}}$$

The main advantage of solving this Lagrange dual instead of the original problem is that the number of variables is greatly reduced [10]. Whereas each element of B was a variable in the original problem, in the dual we have a variable for each column of B. Since columns of B tend to be pretty big (the number of pixels in a basis image), it is a great advantage to be able to reduce them to a single variable. As an example, for bases of size 35×35, this is a reduction by three orders of magnitude in the number of variables!

Learning S: Orthogonal Matching Pursuit

When fixing B, we obtain the optimization problem:

$$\min_S \; ||X - BS||_2^2 + \lambda \sum_j \phi(\vec{s}_j)$$

It is more handy to optimize each column of S separately, and thus instead we'd like to solve the following problem for each $\vec{s}_j$:

$$\min_{\vec{s}_j} \; ||\vec{x}_j - B\vec{s}_j||_2^2 + \lambda\phi(\vec{s}_j)$$

This formulation is more convenient since we get a classical "ϕ-regularized" least squares problem which, for particular ϕs, is well studied in the literature. Our first attempt was to choose the ℓ-1 norm for ϕ and solve the resulting problem with a feature-sign search algorithm proposed in [10]. This algorithm turned out to have a lot of hidden corner cases that were hard to account for in the implementation and so we decided to try something else.

Instead, we chose to use the ℓ-0 norm for ϕ and impose the number of non-zero coefficients k to get the following problem:

$$\min_{\vec{s}} \ ||\vec{x} - B\vec{s}||_2^2, \text{ subject to } ||\vec{s}||_0 \leq k$$

(We dropped the j indices for notational convenience.) Imposing k might seem like an arbitrary restriction, but it allows us to use greedy algorithms that are a bit more efficient, and in practice, as long as we don't choose an excessively small k, it doesn't really affect the results significantly. In fact, under some conditions, the algorithm we used can recover exact sparse $\vec{s}$ (exact in the sense that $\vec{x} = B\vec{s}$) [13].

The algorithm we decided to use is Orthogonal Matching Pursuit (OMP) [12, 11], which has the advantages of being relatively simple and efficient (when correctly implemented). The main idea is to choose the k vectors in B with the highest correlation with $\vec{x}$, and set the corresponding coefficients in $\vec{s}$ to their orthogonal projection on $\vec{x}$. Here are the basic steps of the algorithm:

1. Initialize the residual vector $\vec{r}$, the active set, the coefficients and the loop counter:

$$\vec{r} \leftarrow \vec{x}, \ \text{active} \leftarrow \emptyset, \ \vec{s} \leftarrow \vec{0}, \ j \leftarrow 1$$

2. Find the unused column of B most correlated with the residual:

$$i' \leftarrow \arg\max_{i \notin \text{active}} |\langle \vec{b}_i, \vec{r}\ \rangle|$$

3. Update the active set: $\text{active} \leftarrow \text{active} \cup \{i'\}$.

4. Update the residual vector:

$$\vec{r} \leftarrow (I - B_{\text{active}}(B_{\text{active}}^{\text{T}} B_{\text{active}})^{-1} B_{\text{active}}^{\text{T}})\vec{x}$$

   (B_{active} is the matrix with columns $\vec{b}_j$ for $j \in$ active.)

5. Update the coefficients:

$$\vec{s}_{\text{active}} \leftarrow (B_{\text{active}}^{\text{T}} B_{\text{active}})^{-1} B_{\text{active}}^{\text{T}} \vec{x}$$

   (Similar to above, $\vec{s}_{\text{active}}$ are the elements s_j of $\vec{s}$ for which j is in the active set.)

6. Update the loop counter: $j \leftarrow j + 1$. If it's over k, return the sparse coefficients $\vec{s}$, else go back to step 2.

2.2 Digit Classification

Once the dataset is encoded using the sparse basis, we tried 2 simple classification algorithms, quadratic discriminant analysis and k-nearest neighbours. Both receive the labelled training data and the unlabelled test data as input and output the labels for the test data.

Quadratic Discriminant Analysis

Quadratic discriminant analysis fits multivariate normal distributions on the data. It then classifies the test data points by computing the likelihood of belonging to any group and assigns it to the most likely. QDA fits the normal distribution using the covariance matrix which is calculated on large dataset with the following equation.

$$C_i = \frac{1}{\sum_{i=1}^{k} n_i} Q_i Q_i^T$$

$$Q_i = [\ (\vec{x}_j - \vec{\mu}_i) \quad \dots\ ] \qquad \forall j \textit{ where } \vec{x}_j \textit{ belongs to group } i$$

Where k is the number of groups, n_i is the size of group i and C_i is the covariance matrix for group i calculated using the data points belonging to i.

k-nearest Neighbour

k-nearest neighbours (knn) classifies a data point $\vec{x}_i$ by finding the k nearest $\vec{x}_i$ with respect to some distance function (euclidean in our case) and setting the label to the most seen value in the k-nearest neighbours of $\vec{x}_i$.

2.3 Implementation

The implementation was carried out in MATLAB [14]. To solve the Lagrange dual, we used an off-the-shelf function optimizer, `fminunc` from the Optimization Toolbox [15], which uses a subspace trust-region method. The OMP implementation and the wrapper putting everything together was coded by us. The classification part was done using tools from the MATLAB Statistics Toolbox [16]. Namely, we used the function `classify` for QDA and `knnclassify` for k-nearest neighbours.

3 Results

We ran the algorithm described above on the full MNIST database for 100 iterations, as the bases were pretty stable at that point. We haven't been able to try much more iterations, as it is a very time-consuming process on desktop machines. We generated two sets of bases:

- One containing 64 images and a limitation of no more than 16 non-zero coefficients for each representation of the training images. These bases are shown in figure 2.
- An other with 144 images and with the same limitation as above on non-zero coefficients. These bases are shown in figure 3.

These parameters were chosen in the hope that we could observe and comment on the differences (if any) that the number of basis would incur on the final results.

The first and most obvious thing we can observe is that some basis look like digits (like faint versions of some of the originals) and that, on the whole, the results do not resemble typical results obtained in sparse coding (like that of figure 1). This will be discussed at length in the next section. In the second set of bases (see figure 3), we can notice that a lot more bases are specialized to specific characters. This is expected since having bases specialized in recognizing recurrent patterns will favour sparse coefficients.

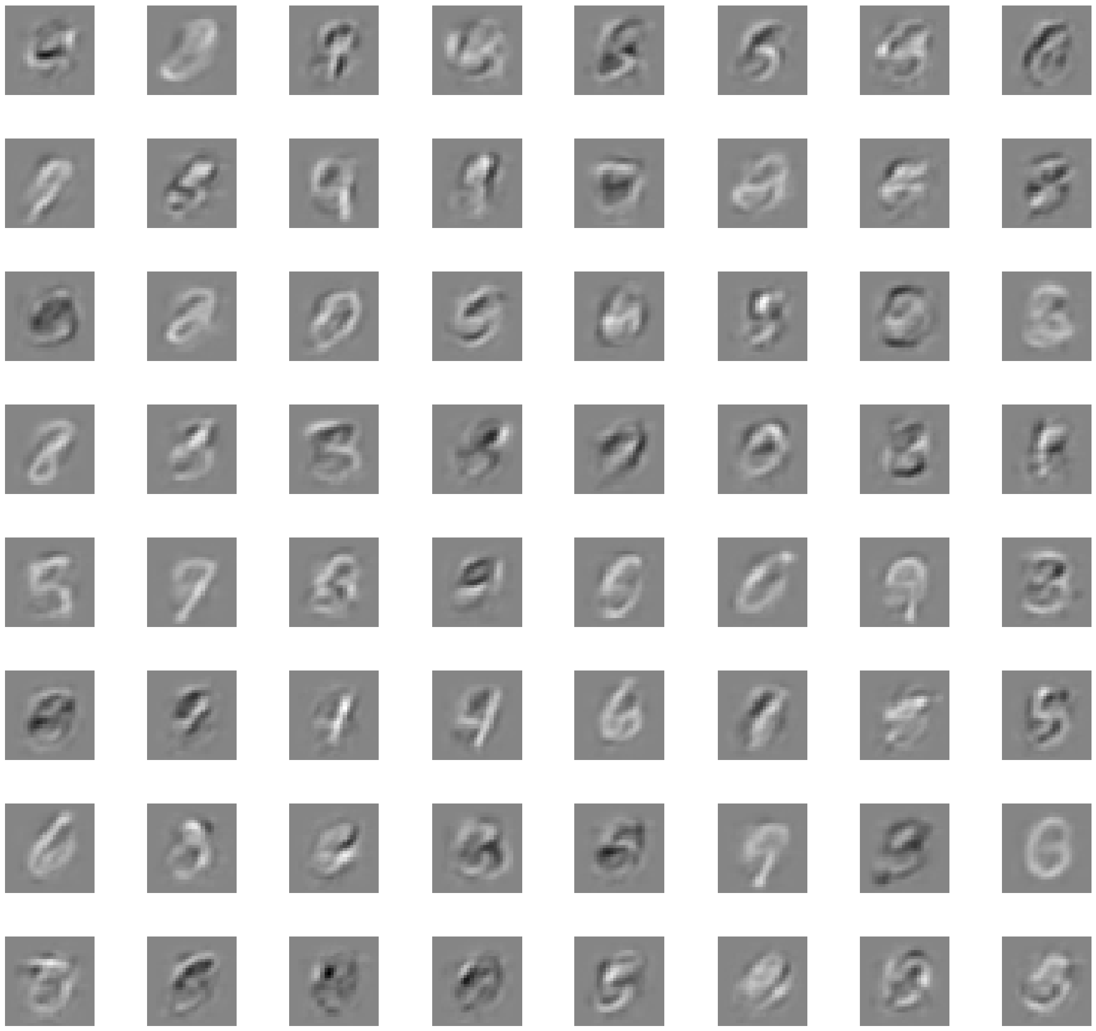

Figure 2: Bases obtained when running the algorithm for 100 iterations and choosing 64 bases.

An interesting way to evaluate the quality of our results is by reconstructing some images from the learned bases and comparing to the originals. Such a comparison can be found in figure 4. As can be observed, the reconstructions are a bit more blurry than the originals but on average they are very close, which is very satisfying. By looking closely, we can see faint shapes and variations in the reconstructions that are not part of the original images. These could be indicative that we did not use enough bases.

Figure 5 shows a small (but pretty representative) sample of digit images that were not classified correctly. We picked them because they best represented the "categories" of wrong classifications we observed. For instance, we can see that digits C3 and D4 are very bold or that D2, C4 and B4 show very exaggerated features. In a similar manner, B1 and A2 have missing components (the 8 is not closed and the lower curve of the 5 is barely visible) and A1 and A4 are heavily tilted. It is easy to see why a computer (or our algorithm in particular) would not classify these types of images correctly (and we will discuss that further in the next section), but some images appear to clearly represent their digits (D1, D3, B3 and A3). It is interesting that our algorithm can't classify those images as well.

As for the classification part, table 1 shows the error rates we obtained. The table also shows results obtained by others (with different techniques) for comparison purposes. As notation, in the table, bases #1 is the one of figure 2 and #2 is shown in figure 3.

In general, QDA performed better. Current methods (mainly based on convolution nets)[19] achieve significantly better results than our approach. Nevertheless we achieved reasonably good results, comparable to the older algorithms, which tells us that our approach is valid.

Algorithm	Error Rate (%)
QDA - bases #1	6.15
QDA - bases #2	8.84
k-nearest neighbours - bases #1	7.68
k-nearest neighbours - bases #2	13.43
linear classifier (1-layer NN)[18]	12.0
k-nearest neighbors, Euclidean distance (L2)[18]	5.0
large convolution net, unsupervised pretraining [19]	0.53

Table 1: Classification results obtained using our algorithms and others'. Here, k=3.

4 Discussion

4.1 Results Analysis

Sparse Basis

It is important to note that our resulting bases don't look like edge detectors like we claimed they should. This is because the edge detectors come from sparse bases generated from patches of natural images. Natural images have many complex dependencies which the sparse bases are trying to capture. In the simple domain of handwritten character, the sparse bases can do much better than edge detectors and generate more significant bases for this dataset. This is why we observe many bases that are similar in part or entirely to handwritten digits. Some digits appear more than others. We believe that this is because some digits have various way of being written but vary little within that style so it could be efficient to dedicate a whole basis to the recurrent shapes.

LDA vs QDA

We first tried linear discriminant analysis which computes a pooled covariance matrix instead of computing one for every group. It can be useful to do this when you can assume that all groups have similar variance and when you have little data to use to compute the covariance matrix as this allows us to use the data of all groups for one covariance matrix. Though, in this task, we have observed large differences in the variance of various characters which meant that QDA was much more successful. A quick example of the differences in variance would be to compare how much variance there is in the way people write the number one compared to how people write the number five.

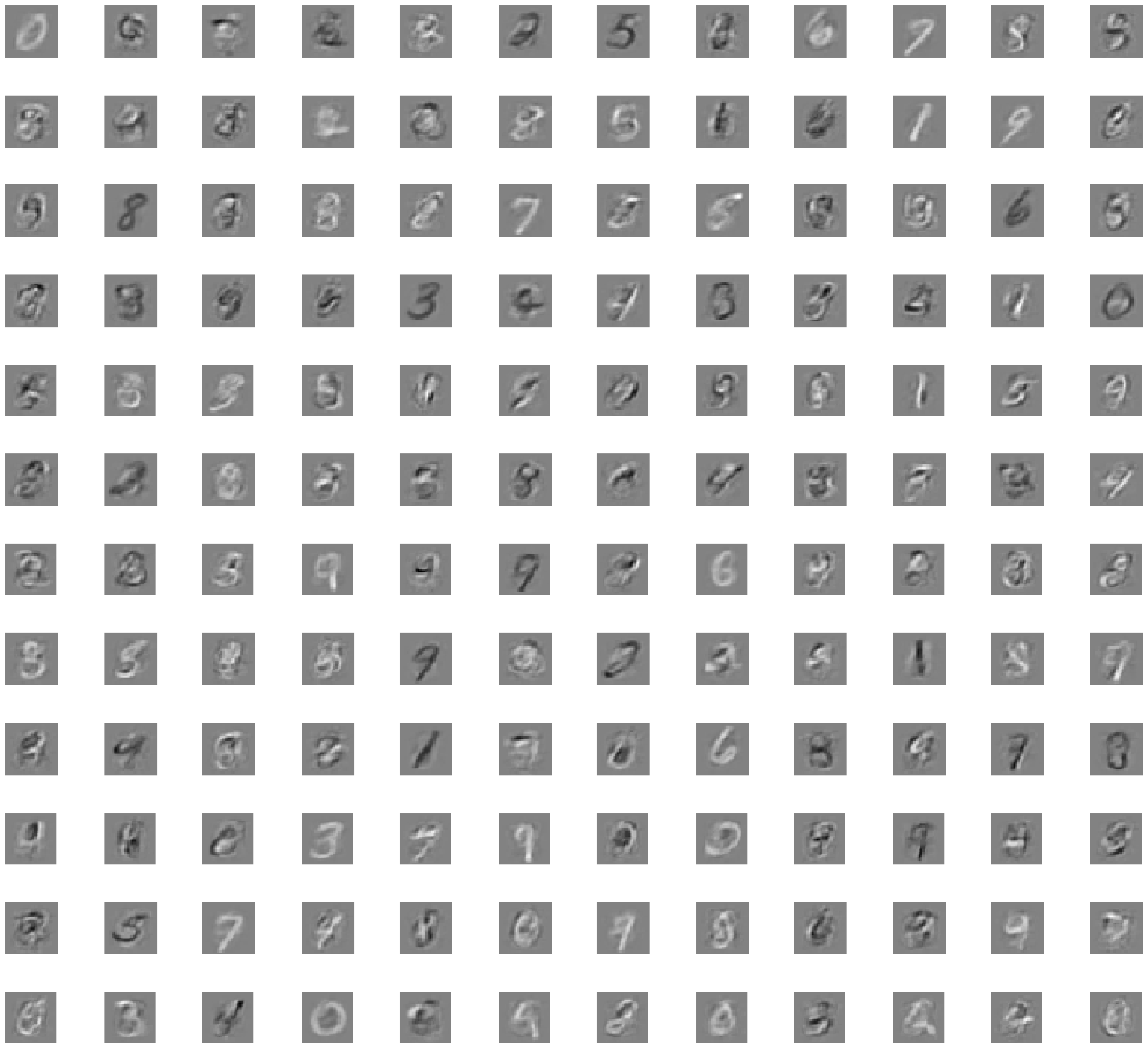

Figure 3: Bases obtained when running the algorithm for 100 iterations and choosing 144 bases.

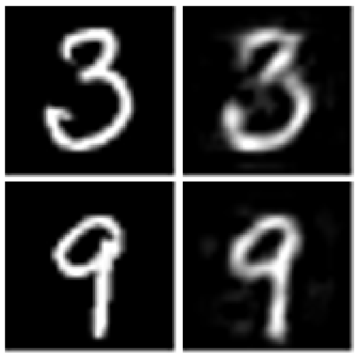

Figure 4: A comparison between two random digits directly from the database (on the left) and their reconstructions from the learned bases and coefficients (on the right). These reconstructions were obtained from the bases in figure 2.

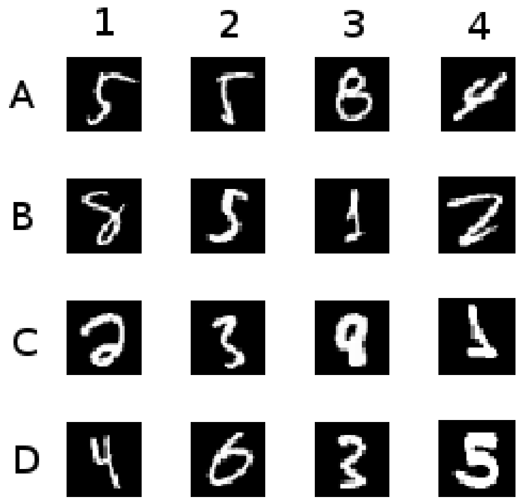

Figure 5: A sample of digit images that were misclassified by our techniques. The rows and columns have been labelled for ease of reference.

4.2 Possible Improvements

Sparse coding

We can improve our sparse bases by using a different algorithm to generate them. OMP approximates the problem of finding the most sparse bases. Maybe we could observe a notable improvement if we solved this problem exactly. Also, since bases take a lot of computation time, we weren't able to test out various limits on the number of non-zero coefficients, meaning that we might of been able to improve our bases without changing our approach.

Classification

We assumed, for the classification, that the basis coefficients of similar characters are going to be close together using an euclidean distance function. Even thought this approach gave good results, the assumption of clustering of the character might be a little strong. Given more time, we could of tried some more powerful algorithms (e.g. neural-net, SVM) that would of allowed us to capture the more subtle relations between the coefficients. Another thing to note is that none of the algorithms we used took advantage of the sparseness of the representations. There may be ways to exploit this to improve the quality of the results (or at least the performance of the algorithms).

References

[1] Hubel D., Wiesel T. (1979), *Brain Mechanisms of Vision*, Scientific American, 241: 150-162.

[2] Olshausen B. A., Field D. J. (1997), *Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?*, Vision Research, 37: 3311-3325.

[3] Olshausen B. A., Field D. J. (1996), *Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images*, Nature, 381: 607-609.

[4] Field D. J. (1993), *Scale-invariance and self-similar "wavelet" transforms: an analysis of natural scenes and mammalian visual systems*, Wavelets, Fractals, and Fourier transforms (pp. 151-193), Oxford: Oxford University Press.

[5] Kavukcuoglu K., Lecun Y. (2008), *Fast Inference in Sparse Coding Algorithms with Applications to Object Recognition*, Technical Report, Computational and Biological Learning Lab, Courant Institute, NYU.

[6] Baum E. B., Moody J., Wilczek, F (1988), *Internal representations for associative memory*, Biological Cybernetics, 59: 217-228.

[7] Foldiak P. (1995), *Sparse coding in the primate cortex*, The handbook of brain theory and neural networks (pp. 895-989), Cambridge, Massachusetts: MIT Press.

[8] Baddeley R. (1996), *An efficient code in V1?*, Nature, 381: 560-561.

[9] Lewicki M. S., Sejnowski T. J., Hugues H. (1998), *Learning Overcomplete Representations*, Neural Computation, 12: 337-365.

[10] Honglak L., Battle A., Raina R., Ng A. Y. (2007), *Efficient sparse coding algorithms*, Advances in Neural Information Processing Systems (NIPS), 19: 801-808.

[11] Bach F., Mairal J., Ponce J., Sapiro G. (2009), *Sparse Coding and Dictionary Learning for Image Analysis*, tutorial given at International Conference on Computer Vision (ICCV).

[12] Cai T. T., Wang L. (2011), *Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise*, IEEE Transactions on Information Theory, 57: 1-26.

[13] Tropp J. A. (2004), *Greed is good: Algorithmic results for sparse approximation*, IEEE Transactions on Information Theory, 50: 2231-2242.

[14] MATLAB version 7.12. Natick, Massachusetts: The Mathworks Inc., 2011.

[15] MATLAB Optimization Toolbox version 6.0. Natick, Massachusetts: The Mathworks Inc., 2011.

[16] MATLAB Statistics Toolbox version 7.5. Natick, Massachusetts: The Mathworks Inc., 2011.

[17] LeCun Y., Cortes C., *The MNIST Database*, `http://yann.lecun.com/exdb/mnist/`, retrieved 2012-04-23.

[18] LeCun Y., Bottou L., Bengio Y., Haffner P. (1998), *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86: 2278-2324.

[19] Jarrett K., Kavukcuoglu K., Ranzato M., LeCun Y. (2009), *What is the Best Multi-Stage Architecture for Object Recognition?*, Proceedings of International Conference on Computer Vision (ICCV).