Dictionary Learning Library for MATLAB (v2)

Adam S. Charles

May 15, 2015

1 Introduction

Sparse coding is a concept that is currently obtaining the state of the art results in many fields, including image processing and theoretical neuroscience [4, 6]. The underlying concept of sparse coding is that a high dimensional data can be represented by very few coefficients in some dictionary. Thus although a vector $\boldsymbol{x}$ may exist in $\mathbb{R}^n$, it may lie on a manifold of dimension $m \ll n$. $\boldsymbol{x}$ can thus be written as

$$\boldsymbol{x} = \mathcal{D}\boldsymbol{a} + \epsilon \tag{1}$$

where $\mathcal{D}$ is the dictionary (each column is an element of the dictionary), $\boldsymbol{a}$ is the sparse set of coefficients which generate $\boldsymbol{x}$, and ϵ is a white Gaussian noise term indicating imperfections in the generative model.

Two main questions arise in the sparse coding framework with respect to Equation (1):

1. Given a data point $\boldsymbol{x}$ and a dictionary $\mathcal{D}$, how can I recover the coefficients $\boldsymbol{a}$?

2. Given a set of data points $\{\boldsymbol{x}_k\}_{k\in[1,K]}$, can I learn both the coefficients that generated then as well as the dictionary $\mathcal{D}$ in which the decompositions are sparse?

The answer to the first of these questions is well studies in the field of Compressed Sensing, with many readily available packages available [3, 5]. The answer to the second question, however, comes in two distinct flavors, each stemming from a slightly different interpretation of Equation (1). In one interpretation, the Compressed Sensing approach is taken and the answer comes in the form of the K-SVD algorithm [1]. In a different interpretation, Equation (1) is viewed as a probabilistic model, where a MAP estimation is used to determine $\boldsymbol{a}$. In this case, a highly kurtotic prior is placed on $\boldsymbol{a}$, and the dictionary can be learned in a statistical manner, by sampling from a large number of data samples [7]. This document is for the dictionary learning library version 1.0, which performs dictionary learning over a dataset using the statistical unsupervised method in [7].

2 Functionality

2.1 Theory of Dictionary Learning

The statistical method of learning dictionaries for sparse coding is based on maximizing the probability distribution over $\mathbf{x}$, or equivalently minimizing an energy function. In this method, Equation (1) is used to write a likelihood probability distribution on $\mathbf{x}$,

$$p(\boldsymbol{x}|\boldsymbol{a},\mathcal{D}) = Ze^{-\frac{\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2}{2\sigma_\epsilon^2}} \tag{2}$$

where σ_ϵ is the variance of the noise term ϵ, and Z is the normalizing constant for the distribution. Now a prior is placed on the coefficients $\boldsymbol{a}$. The prior distribution is necessarily highly kurtotic, and while [7] uses a Cauchy distribution the most used currently is the Laplacian distribution. This distribution is chosen because of it's relationship to the ℓ^1 norm used in compressed sensing coefficient recovery. The posterior distribution is then

$$\begin{align} p(\boldsymbol{a}|\boldsymbol{x},\mathcal{D}) &\propto p(\boldsymbol{x}|\boldsymbol{a},\mathcal{D})p(\boldsymbol{a}|\mathcal{D}) \tag{3} \\ &\propto e^{-\frac{\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2}{2\sigma_\epsilon^2}} e^{-\frac{\sqrt{2}}{\sigma_a}\|\boldsymbol{a}\|_1} \tag{4} \end{align}$$

where σ_a^2 is the variance of the Laplacian distribution on the coefficients. In Equations (3) and (4), the constant scaling factors are dropped since they do not effect the arg max of the posterior distribution. The conditional in the prior distribution is purely to maintain the concept that this calculation is valid given a dictionary, since technically the prior is independent of $\mathcal{D}$. Solving the MAP inference problem yields the coefficients $\boldsymbol{a}$, given a dictionary $\mathcal{D}$:

$$\begin{align} \hat{\boldsymbol{a}} &= \arg\max_{\boldsymbol{a}} (p(\boldsymbol{a}|\boldsymbol{x},\mathcal{D}) \tag{5} \\ &= \arg\max_{\boldsymbol{a}} \left(e^{-\frac{\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2}{2\sigma_\epsilon^2}} e^{-\frac{\sqrt{2}}{\sigma_a}\|\boldsymbol{a}\|_1} \right) \tag{6} \\ &= \arg\min_{\boldsymbol{a}} \left(\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2 + \lambda\|\boldsymbol{a}\|_1 \right) \tag{7} \end{align}$$

where $\lambda = 2\sqrt{2}\sigma_\epsilon^2/\sigma_a$. Minimizing the cost function in Equation (7) (the negative log of the posterior) with respect to $\boldsymbol{a}$ is the optimization given $\mathcal{D}$, and is a well studied convex optimization problem. In dictionary learning, however, the same energy function needs to be minimized with respect to $\mathcal{D}$ as well.

Finding $\mathcal{D}$ can be viewed as either a maximum likelihood (ML) estimate or another MAP estimate. In the ML version, the optimization is

$$\arg\min_{\mathcal{D}} p(\boldsymbol{x}|\mathcal{D}) = \arg\min_{\mathcal{D}} \int_{\mathbb{R}^p} p(\boldsymbol{x}|\boldsymbol{a},\mathcal{D})p(\boldsymbol{a})d\boldsymbol{a} \tag{8}$$

Which required sampling from the posterior. In [7], Olshausen and Field show that the distribution is tight about the maximum peak $\hat{\boldsymbol{a}}$, thus the integral can be estimated by finding the maximum and the optimization becomes:

$$\int_{\mathbb{R}^p} p(\boldsymbol{x}|\boldsymbol{a},\mathcal{D})d\boldsymbol{a} \approx \langle p(\boldsymbol{x}|\mathcal{D},\hat{\boldsymbol{a}})\rangle \tag{9}$$

To minimize this likelihood, a gradient descent algorithm can be used with steps on the i^{th} dictionary element (columns of $\mathcal{D}$) given by

$$\Delta\mathcal{D}_i \propto \langle \hat{\boldsymbol{a}}(\boldsymbol{x} - \mathcal{D}\hat{\boldsymbol{a}})\rangle \tag{10}$$

where $\langle\rangle$ denotes an average over some sample set of the data.

In order to minimize the likelihood, the following algorithm was set up:

1. Initialize $\mathcal{D}$ to some random dictionary
2. pick a random subsample of the data
3. Find $\boldsymbol{a}$ given $\mathcal{D}$ for each $\boldsymbol{x}$ chosen
4. Take a gradient step on $\mathcal{D}$ given by Equation (10)
5. Repeat steps 2 - 4 until convergence

The other method to optimize a dictionary places a prior over the elements of $\mathcal{D}$ as well as $\boldsymbol{a}$, resulting in the expended posterior

$$p(\boldsymbol{a},\mathcal{D}|\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{a},\mathcal{D})p(\boldsymbol{a}|\mathcal{D})p(\mathcal{D}) \tag{11}$$

$$= e^{-\frac{\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2}{2\sigma_\epsilon^2}} e^{-\frac{\sqrt{2}}{\sigma_a}\|\boldsymbol{a}\|_1} e^{-\frac{\|\mathcal{D}\|_F}{2\sigma_\mathcal{D}^2}} \tag{12}$$

In this case the elements of $\mathcal{D}$ are *i.i.d.* Gaussian random variables with zero mean and variance $\sigma_\mathcal{D}^2$. The MAP inference is then a joint inference problem given by

$$\{\hat{\mathcal{D}},\hat{\boldsymbol{a}}\} = \arg\min_{\{\mathcal{D},\boldsymbol{a}\}} \left(\|\boldsymbol{x}-\mathcal{D}\boldsymbol{a}\|_2^2 + \lambda\|\boldsymbol{a}\|_1 + \lambda_2\|\mathcal{D}\|_F\right) \tag{13}$$

where $\lambda_2 = \sigma_\epsilon^2/\sigma_\mathcal{D}^2$. The learning procedure (in performing an alternating minimization as before) is essentially the same, but with an extra term to the gradient descent step:

$$\Delta\mathcal{D}_i \propto \langle \hat{\boldsymbol{a}}(\boldsymbol{x} - \mathcal{D}\hat{\boldsymbol{a}}) - 2\lambda_2\mathcal{D}_i\rangle \tag{14}$$

2.2 Code Functionality

The code in the dictionary learning library includes code to learn a dictionary, $\mathcal{D}$, for data in the form of vectors, image patches, or data cubes (e.g. videos) sing either a likelihood or MAP estimate on $\mathcal{D}$. The code is set to be able to use a preset conjugate gradient descent algorithm to infer the coefficients, but any inference algorithm can be used, with the appropriate wrapper (see Section 2.2.2). Some wrappers are included for some popular inference methods, such as `l1_ls` [5]. For general help with a specific function, type help then the function name for comments on its use.

Table 1: Functions included in the Dctionary Learning Library

Function Name	Description
`dictionary_learn_script`	Example script for setting up dictionary learning on natural images
`learn_dictionary`	Main function to learn a dictionary, using parfor parallelization
`initialize_dictionary`	Function to initialize a dictionary
`gen_multi_infer`	Function to infer coefficients for a data sampling in parallel
`dictionary_update`	Function to update a dictionary using either a ML or MAP method
`cg_l2l1_wrapper`	Wrapper for cg l2l1 (included)
`groupLCA_wrapper`	Wrapper for group-sparse inference via the LCA (included)
`rwLCA_wrapper`	Wrapper for re-weighted ℓ_1 inverence via the LCA (included)
`l1ls_wrapper`	Wrapper for l1 ls (http://www.stanford.edu/ boyd/l1 ls/)
`l1ls_nneg_wrapper`	Wrapper for l1 ls with non-negativity constraints (in l1 ls package)
`SolveMP_wrapper`	Wrapper for Matching Pursuit (MP) (http://sparselab.stanford.edu)
`SolveOOMP_wrapper`	Wrapper for Optimized Orthogonal MP (http://sparselab.stanford.edu)
`perform_omp_wrapper`	Wrapper for regular OMP (http://www.personal.soton.ac.uk/tb1m08/sparsify/sparsify.html)
`greed_omp_qr_wrapper`	Wrapper for QR-OMP (http://www.personal.soton.ac.uk/tb1m08/sparsify/sparsify.html)
`cg_l2l1`	Function for sparsity-based inference via conjugate gradient descent
`groupLCA`	Function for group-sparse inference via the LCA
`rwLCA`	Function for re-weighted sparse inference via the LCA
`mintotal`	Function used in cg l2l1 for the optimization
`basis2image2`	Function to reshape a 2D dictionary into a single plot-able image
`dict_plot1d`	Function to plot a vector type dictionary
`learn_dictionary spmd`	Outdated function to learn a dictionary, using spmd parallelization

2.2.1 Included Functions

The included functions are shown in Table 1. The main functions are `learn_dictionary` and `learn_dictionary_spmd`. These functions are equivalent in terms of the input/output characteristics, with the difference being the change in the parallelization scheme.

In `learn_dictionary`, the interior loop where the coefficients of each selected data point is inferred is parallelized using a parfor loop. In `learn_dictionary_spmd`, the parallelization is accomplished by using single program multiple data (spmd) parallelization. The main difference is that parfor apportions the data to the workers as it is selected, while spmd separates the whole data set at the start of the program, and sends only commands to the workers. Different parallelization schemes will be more efficient based on the computer type, data dimensionality and dictionary size. The script `dictionary_learn_script` is an example of starting a `MATLAB` worker pool, loading data, initializing a dictionary and learning the dictionary for a set of natural images. The file IMAGES.mat referenced can be found on Dr. Olshausens website: http://redwood.berkeley.edu/bruno/sparsenet/. The full list of options supported by the included code is:

- `opts.data_type` - Type of data ('vector, 'square or 'cube)

- `opts.grad_type` - Type of cost function for gradient descent: ‘norm or ‘frob.
- `opts.sparse_type` - Type of inference to use. For full set of options, see `multi_infer.m`
- `opts.n_elem` - Number of dictionary elements
- `opts.in_iter` - Number of samples per iteration
- `opts.iters` - Number of iterations to run the algorithm for
- `opts.GD_iters` - Number of gradient descent steps per iteration
- `opts.verb` - 1 if verbose outputs are desired
- `opts.bl size` - Block size for ‘square or ‘cube data types
- `opts.dep_size` - Depth size for ‘cube data type
- `opts.nneg_dict` - Choose non-negative values only for the dictionary
- `opts.step_size` - Initial step size for gradient descent
- `opts.decay` - Rate of decay for gradient descent step size
- `opts.lambda` - Lambda value for l1-regulated inference schemes
- `opts.lambda2` - Second lambda value for ‘frob energy function
- `opts.tol` - Tolerance for inference schemes
- `opts.ssim_flag` - Choose weather to normalize the data pre-inference
- `opts.std_min` - Minimum standard deviation (use with `.ssim_flag`)
- `opts.save_every` - Number of iterations to save after
- `opts.save_name` - Filename to save the dictionary in
- `opts.bshow` - 0 to not plot intermediary dictionaries. Else plot every `opts.bshow` iterations.
- `opts.disp_size` - Dimensions of figure to display the dictionary

Most of these options have default values and do not need to be set. The bare minimum for running the dictionary learning function is the type of data, the number of dictionary elements, the block size (‘square or ‘cube data) and depth (‘cube data). This dictionary learning package is set up to allow for any inference scheme to be used to find the sparse coefficients at each iteration. All that needs to be used is an appropriate wrapper. A number of wrappers for useful inference schemes have been provided, with the functions available at the corresponding websites in Table 1. For more specific details on the other functions, simply use the help command in MATLAB.

2.2.2 Writing Your Own Wrapper

This package includes a number of wrappers for solving the internal inference problem. In addition to wrappers for standard code packages that solve the sparse inference problem, additional wrappers for alternate inference procedures are available. Specifically, the functions for solving re-weighted ℓ_1 and group-sparse inference, based on the locally competitive algorithm (LCA) [2] are included, along with their wrappers. Additionally, a number of other wrappers which call functions in packages available from other labs (the location of these packages are shown in Table 1. While these wrappers are included for some readily available code packages, any function can be used as long as a wrapper is written for it and passed to the main function. To write a wrapper, simply use the same inputs/outputs as the existing wrappers and any extra parameters needed can be passed through using the `opts` struct. The inputs to the wrapper must be of the form `(dictionary_n, x_im, opts)`, and the output must be a single output: `coef_vals`. The wrapper simply extracts the necessary options from opts and organizes the inputs into the inference function.

References

[1] M. Aharon, M. Elad, A. Bruckstein, and Y. Katz. K-svd: An algorithm for designing of overcomplete dictionaries for sparse representations. *IEEE Proceedings - Special Issue on Applications of Compressive Sensing & Sparse Representation.*

[2] A. S. Charles, P. Garrigues, and C. J. Rozell. A common network architecture efficiently implements a variety of sparsity-based inference problems. *Neural Computation*, 24(12):3317–3339, 2012.

[3] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[4] M. Elad, M.A.T. Figueiredo, and Y. Ma. On the role of sparse and redundant representations in image processing. *IEEE Proceedings - Special Issue on Applications of Compressive Sensing & Sparse Representation*, Oct 2008.

[5] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large scale l1-regularized least squares. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):606–617, Dec 2007.

[6] B. A. Olshausen and D.A. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(13):607–609, June 1996.

[7] B. A. Olshausen and D.A. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997.