Exploiting Sparsity in Supervised Learning

Sharan Vaswani
Department of Computer Science
University of British Columbia
Vancouver,BC,Canada
sharanv@cs.ubc.ca

Abstract

We explore various aspects of sparsity in the context of supervised learning. In particular, we review sparse coding techniques used to exploit sparsity in the input space for a supervised classification task. We empirically evaluate the importance of enforcing sparsity in the weight space through regularization. We use a compressed sensing framework which exploits sparsity in the output space for a multilabel classification problem. We perform experiments to quantify the effect of different sources of sparsity on the classification accuracy and performance. We also outline some promising directions for future work.

1 Introduction

Supervised learning relies on a domain expert who teaches the learning system with necessary supervision. In particular, the expert supervises the learning by providing correct answers i.e. ground truth data (labels in the case of classification and real values in the case of regression) for a set of input instances called the training set. The supervised learning system once trained can then be used to predict labels/values for unseen input instances called the test set. More formally, the training set consists of N data points $D(x_i, y_i)_{i=1}^{N}$ where x_i is the vector which describes the features of the i_{th} example in the training set and y_i is the vector of labels for that particular example. For the classification task, the labels correspond to integer values which represent the class to which the example belongs. The vectors x correspond to the input space whereas the y vectors lie in the output space. In our study, we concentrate on the task of binary multilabel classification i.e. the output space is $\{-1, +1\}^d$ where d is the number of labels. Similarly, let m denote the dimension of the input vectors x_i. Hence the objective is to find a mapping $f : X \to Y$ where $X \in R^{m \times N}$ and $Y \in \{-1, +1\}^{d \times N}$. The mapping f is characterized by a set of parameters which denote the importance of each input feature for the classification task at hand. These parameters called weights lie in what we refer to as the weight space. We explore the sources of sparsity in the input, weight and output spaces.
Input sparsity arises due to correlations between features and/or examples. Due to these correlations, each example x_i resides in a lower dimensional subspace and can be represented by a vector of $m^{'}$ non-zero coefficients where $m^{'} < m$. The input space is thus sparse in a particular set of bases. This set of n basis functions are either fixed (eg: natural images are represented by intensity values but are sparse in the wavelet domain) or can be inferred from the data. This can be achieved by learning a dictionary of n basis functions which capture higher level or more complex features in the data [20] [18]. The dictionary can be overcomplete i.e. $n > m$ or undercomplete $n < m$. Using an undercomplete dictionary represents the same amount of information as the original data in a smaller set of basis functions and can lead to dimensionality reduction in the feature space. This becomes important for applications in computer vision and

natural language processing where the data is high dimensional and has redundancy in the representation. For example the raw pixel representation of an image [14] in vision or the bag of word representation of a document in natural language processing. Learning an overcomplete dictionary is equivalent to identifying a large set of patterns or possible feature combinations in the data. The size of the dictionary depends on the application.
Output sparsity can be exploited in the case of multi-label classification where the output vector y is multi-dimensional. Although the output space has a large dimension, this space is usually sparse i.e there are only a few positive labels for each input example. For example, in vision, the labels may correspond to the presence / absence of a particular object in the image for a large number of objects [12] though a particular image will contain only a few objects and thus is a positive example for a very small subset of labels.
Weight space sparsity is different from input / output sparsity in that it does not arise naturally because of the structure of the problem. Rather it is imposed as a constraint by the learning algorithm. This constraint is referred to as regularization and can be imposed by using an $l1$ penalty on the weight vectors. This ensures that some of the weights will be zero or small. This is referred to as Lasso regression [27]. Introducing an $l2$ penalty on the weights also serves a similar purpose and is as known as Ridge regression [11]. In the Bayesian framework, these techniques can be interpreted as introducing a prior on the weights. Regularization techniques are common and typically used in most machine learning algorithms to prevent overfitting.
In this study, we explore some of the techniques to exploit sparsity to achieve higher classification rates and lower training times. For input sparsity, we empirically evaluate two methods - the sparse coding algorithm by Lee.et.al in [17] and the autoencoder approach by Hinton et.al. [10] to reduce the dimension of the input space. We use a compressed sensing framework described in [12] to account for the sparsity in the output space. We also perform a simple experiment to highlight the effect and importance of regularization. The subsequent paper is structured as follows: Section 2 reviews some of the previous work and section 3 describes the specific algorithms used. We describe our experiments and present our results in 4. We conclude the paper and outline some directions for future work in section 5

2 Related Work

In section 1, we introduced the notion of learning a set of basis functions to represent the input data. This problem of learning a dictionary is known as sparse coding. Sparse coding can also motivated from a neuroscience perspective. Sparse codes for natural images correspond to receptive fields of neurons in the primary visual cortex [19] [21].This makes sparse coding a plausible model of the visual cortex [25]. Sparse coding can be used to improve classification performance by using richer features and also to reduce the training time on large datasets through dimensionality reduction [31] [23] [9] [32]. Several different approaches have been proposed for generating sparse codes to represent data. One method to capture linear correlations between features in the input data is principal components analysis [8](PCA). PCA however has two limitations - the transformation from the original input space to the new basis space is linear. Also we can't learn an overcomplete dictionary using PCA. Other more sophisticated approaches include independent component analysis [15], autoencoder based approach [10] and an alternating optimization proposed in [17]. The sparse codes learnt using these algorithms is usually fed to a classifier to learn an input-output mapping. A recent paper [9] learns the sparse codes and classifier weights simultaneously. Similarly there have been a number of techniques to exploit sparsity in the output space. The standard approach in multilabel classification is the one vs all [24] approach in which each label is treated independently and a separate classifier is trained for each label. However this approach becomes prohibitive as the number of labels becomes large. A common idea is to exploit the structure in the problem [6] [5] [29]. [33] provides a recent survey of the techniques used in multilabel classification. [13] uses group sparsity among labels in the output space whereas [12] uses compressed sensing algorithms for multilabel classification.

3 Algorithms

3.1 Input Space

We implement two approaches for sparse coding - the alternating optimization approach proposed in [17] and the autoencoder based approach described in [10].

3.1.1 Alternating Direction Method of Multipliers

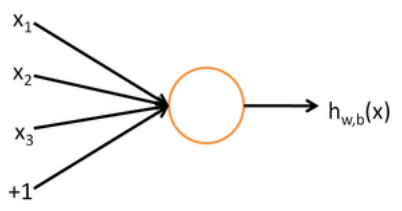
For obtaining a sparse representation of the data, we need to learn a set of basis functions (B) in which the original input data is sparse i.e. we require that only few of the basis coefficients (s) are non-zero for each input vector. Let $B \in R^{m \times m^{'}}$ represent the set of bases which transform the m dimensional input space to $m^{'}$ dimensions where $m^{'}$ may be greater than or less than m depending on whether the dictionary is overcomplete or undercomplete. Let $S \in R^{m^{'} \times N}$ denote the set of coefficients for each of the N data points in the transformed subspace. We desire that the S matrix be sparse. We can achieve this by imposing a penalty ϕ (usually L1) on S. We also introduce a regularization on B. Hence given the data matrix $X \in R^{m \times N}$, our objective function is

$$\begin{aligned} &\min_{B,S} ||X - BS||_F^2 + \beta \sum_{i,j} \phi(S_{i,j}) \\ &s.t. \sum_i B_{i,j}^2 \leq c \ \ \forall j = 1, 2...n \end{aligned} \tag{1}$$

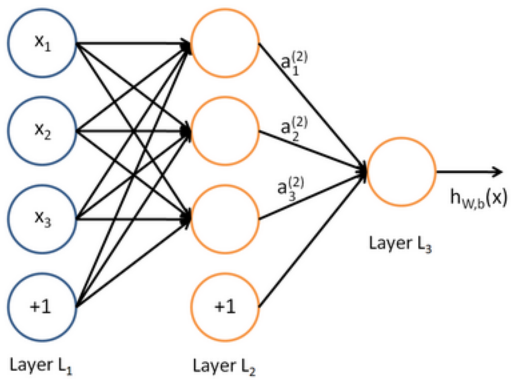
The first term in the objective function is the fidelity constraint whereas the second term enforces sparsity in the coefficients. The constraint imposes regularization on the basis functions. We observe that the objective function is convex with respect to either B or S but is not jointly convex. Hence we use the Alternating Direction Method of Multipliers (ADMM) to alternately minimize the function over S (S subproblem) and B (B subproblem). We iterate between solving the S and B subproblems for each input example. If the penalty is an L1 penalty, the S subproblem is a L1 regularized least squares problem. [17] uses a feature sign search algorithm to guess the sign for each $S_{i,j}$. Once the sign is correctly determined, $|S_{i,j}|$ can be simplified as $S_{i,j}$ or $-S_{i,j}$ depending on whether it is positive or not. The S problem then reduces to an unconstrained quadratic optimization problem and can be solved efficiently. The B subproblem is a least squares problem with quadratic constraints and can be solved efficiently. We convert the constrained optimization problem into the Lagrangian form and solve the resulting problem using gradient descent. After a certain number of iterations, we learn a basis B in which the original input is sparse.

3.1.2 Autoencoder

Neural networks have been used in a variety of classification tasks. A neuron which is the basic unit of a neural network takes a set of inputs and outputs a non-linear transformation of these inputs. The inputs to each neuron in the network are weighted by the weight parameters which are learned from the data during the training phase. The non-linearity also known as the activation function is generally a sigmoid or tanh function of the weighted inputs. Figure 1(a) shows a single neuron. Individual neurons connect together to form a neural network which usually has hierarchical or layered architecture. A typical neural network consists of an input layer, one or more hidden layers and an output layer. Each feature x_i is input to a single input layer neuron. The hidden layers perform some non-linear transformation of the original input. The output layer neurons provide the final output for the learning task. For example, if the task at hand is binary classification, then the output layer can consist of 2 neurons which encode a positive and negative estimate of the label. Figure 1(b) shows a simple neural network. Each input is fed into the neural network which outputs a hypothesis or an estimate for the label of that input. Since we know the true labels for the training set, we can find the error in our estimate by using a suitable loss function (usually squared loss). We use the gradient descent algorithm to modify the weights in the network so that the neural network better estimates the true labels for each input. This method of iteratively refining the weights of the network to minimize the loss function is known as backpropagation. This constitutes the training phase of our algorithm. The



(a)



(b)

Figure 1: (a) Single Neuron: x1,x2,x3 are inputs to the neuron weighted by the weight vector w = (w1,w2,w3). +1 represents a bias input and is essentially a shift origin. It is weighted by the bias term(b). $h_{W,b}(x)$ represents the output of the neuron and depends on the weights and the activation function used.(b) Neural network consisting of an input layer of 3 neurons, an output layer and one hidden layer. The output neuron outputs an estimate or hypothesis corresponding to each input fed into the neural network. The weights between neurons of each layer are learned by an optimization procedure

prediction for a test example is done by a feed-forward pass through the network. An autoencoder is a special kind of neural network where the input is equal to the output. Hence the neural network is trained to find a set of weights so that the input can be reconstructed. We use one or more hidden layers such that the number of hidden neurons is less than the input(and output) neurons. Since we train the neural network to estimate the original input, the hidden layer encodes the information in the original input by using a fewer number of features. This imposes the sparsity constraint which enables us to learn sparse codes for the input data. This helps us achieve dimensionality reduction. Each hidden neuron is a dictionary element or basis function and its output value for an input is the corresponding coefficient. If the number of hidden neurons is greater than the input, it is equivalent to learning an overcomplete dictionary to represent the input. We add an explicit sparsity constraint - that the average activation of the hidden neurons is less than a tunable parameter ρ. Thus, we want each neuron to output large values only for a small subset of inputs. Thus, only a few neurons fire for any given input and we have a sparse representation of this input with a majority number of neurons giving small or zero activations(coefficients). We also introduce a regularization term λ on the weights. The final optimization function can be written as:

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}(\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)})||^2)\right] + \frac{\lambda}{2}||W||^2 + \sum_{j=1}^{s} KL(\rho||\rho_j) \tag{2}$$

where W is the set of weights and b is the set of bias units. The first term models how well the hypothesis ($h_{W,b}(x^{(i)})$) fits the true data (y_i) for each of the m examples. The second term is the regularization term on all the weights in the network and helps prevent overfitting. The third term models the explicit sparsity constraint and penalizes a hidden neuron j if its average activation ρ_j is more than ρ. This penalty is modelled as the KL-divergence between two Bernoulli distributions with parameters ρ and ρ_j. This unconstrained optimization problem is typically solved using gradient descent or L-BFGS.

3.2 Output Space

We consider the binary multilabel classification problem as was described in section 1. The output space is $\{-1,+1\}^d$. This implies that if all the labels are independent, we need to solve d binary classification problems following the one vs all classification scheme described in section 2. However in most cases of interest, the output space is sparse although there need not be any specific relation among the multiple labels. Since this is quite a generic assumption which does not depend on the dataset, we use the method described in [12] to exploit this sparsity in the output space. We transform the labels into a $d^{'}$ dimensional space where $d^{'} < d$. Given a label vector y_i, we transform it using an under-determined matrix $A \in R^{d^{'} \times d}$. The number of rows of A i.e. $d^{'}$ determines the amount of compression which depends on the inherent sparsity in the output space. If the output space has a high sparsity, we can compress it such that $d^{'} << d$ without losing much information. Then we just use the one vs all technique on this low dimensional space and train a separate regression model for each of the $d^{'}$ labels. We use the weights learned to predict the $d^{'}$ labels for each of the examples in the test set. Once we obtain the test data predictions in the low dimensional label space, we can use existing compressive sensing algorithms like orthogonal matching pursuit (OMP). These algorithms are able to recover the sparse original vectors y under some conditions dependent on the sparsity of y and properties of the A matrix. We refer the interested reader to [7] [3] [4] for technical constraints on the signal sparsity and properties of the measurement matrix A. [28] explains how signal can be recovered using the orthognal matching pursuit algorithm. Figure 2 graphically explains the algorithm described above.

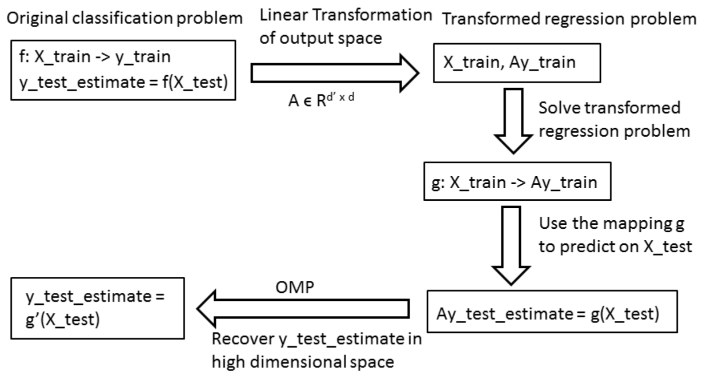
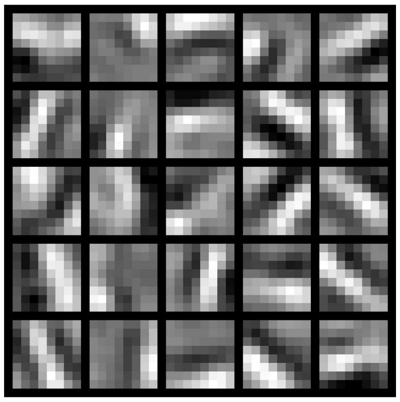

Original classification problem
f: X_train -> y_train
y_test_estimate = f(X_test)
Linear Transformation of output space
A ϵ R^{d' x d}
Transformed regression problem
X_train, Ay_train
Solve transformed regression problem
g: X_train -> Ay_train
Use the mapping g to predict on X_test
Ay_test_estimate = g(X_test)
OMP
Recover y_test_estimate in high dimensional space
y_test_estimate = g'(X_test)

Figure 2: From left to right: the first box represents the original problem i.e we need to find a mapping f from the train data to train labels. We use this mapping to make predictions on the test data. We use the under-determined matrix A to transform the original problem to a regression problem in a low dimensional output space. We solve this problem in this low dimensional label space and use the mapping g to make predictions $Ay\_estimate$ on the test data.We transform these low dimensional estimates to estimates in the original space using compressive sensing algorithms like OMP. Finally, we use a thresholding operation to convert the reconstructed real values to binary values.

4 Experiments and Results

4.1 Input space

We implement the sparse coding algorithms described in section 3. As a sanity check to verify the correctness and quality of our sparse codes, we used the two algorithms to obtain sparse codes for natural images. As was stated in section 2, the sparse codes for natural images correspond to receptive fields of neurons in the primary visual cortex. The receptive fields correspond to bars in different orientations with a centre surround [21] property. Figure 3(a) visualizes the coefficients for a subset of bases learned by the autoencoder algorithm. The results for the ADMM algorithm are similar. This correctly matches the neuron receptive fields. To test the effect of sparse coding on classification performance, we use the MNIST dataset [1] for our experiments. The MNIST dataset consists of 70k 28×28 images of handwritten digits. We use 60k images as our train data and 10k images for testing. We obtain sparse codes for the raw input data using both the ADMM and autoencoder based approach. We learn an undercomplete dictionary and fix the size of the basis set to be 196 for both the algorithms. For the ADMM algorithm, we use 100 iterations of alternating minimization and use the L1 penalty to enforce sparsity in the basis coefficients. We tune the parameter β to 0.4 to obtain through crossvalidation. For the autoencoder, we set the number


(a)

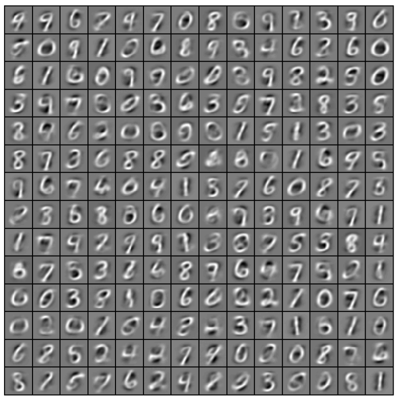

(b)

Figure 3: Coefficients for a subset of bases learned using the autoencoder based approach for (a)natural images and (b) MNIST digits.

of hidden neurons to be 196 and tune the sparsity parameter(ρ) to 0.1 and the weight decay parameter λ to 3×10^{-3}. We use 400 iterations of L-BFGS for training. Figure 3(b) shows a visualization of the coefficients learned for the MNIST digits. The bases in this case correspond to various penstrokes from which different digits are composed. We use a 10 way (each class corresponds to a particular digit) softmax classifier (a multiclass extension of logistic regression). Table 1 gives the out of sample test accuracies for each of the approaches. Thus we observe that a transformation of the original data exploiting the sparsity

Input to softmax classifier	Size of input	Training time	Accuracy
Raw Pixels	784	52.06 s	92.64
Sparse codes from ADMM	196	9.59 s	96.23
Sparse codes from Autoencoder	196	13.43 s	96.63

Table 1: Effect of sparse coding on classifier performance.

can lead to better classification rates and lower training times. A number of state of the art classifiers [16] [22] use sparse coding (in the form of unsupervised pre-training) to transform the input in a similar way.

4.2 Output space

We implement and evaluate the compressed sensing based multilabel classification algorithm described in the previous section. In particular, we use two standard multilabel classification datasets - mediamill and delicious for evaluating our performance. Table 2 describes the characteristics of the two datasets. We

Dataset	#(Instances)	#(Nominal Attributes)	#(Numeric Attributes)	#(Labels)	Cardinality
Mediamill	43907	0	120	101	4.376
Delicious	16105	500	0	983	19.020

Table 2: Multilabel classification dataset characteristics. Mediamill dataset corresponds to 101 semantic concepts in video data [26]. Delicious dataset [30] is extracted from the del.icio.us website and consists of textual data of webpages along with their tags. Cardinality here corresponds to the average number of positive labels for an input example.

do not perform any dimensionality reduction on the data although one can use PCA or the sparse coding approaches described above to achieve this. We use logistic regression to solve the original classification problem in a one vs all manner described in the previous section. Linear regression is used to solve the problem in the transformed low dimensional space. We use orthogonal matching pursuit (OMP) to recover the original output space. For recovering the true labels, OMP requires as input an estimate of the sparsity(k) in the original labels. We estimate the sparsity in the training output space and vary k in its neighbourhood for the test set. We also vary the number of projections $d^{'}$ i.e. the rate of compression of the output space. Higher number of projections will lead to better recovery of labels but we need to solve a greater number of linear regression problems and the training time will increase. Hence we trade off speed and accuracy by varying $d^{'}$. We compare the training and prediction times and the classification performance of the two approaches on both the datasets. Table 3 summarizes these results.

Dataset	Training time(O)	Accuracy(O)	Training time(C)	Reconstruction Time	Accuracy(C)
Mediamill	111.28 s	96.88 % (MSE = 3.11)	20.90 s	53.42 s	95.88 % (MSE = 4.11)
Delicious	144.80 s	99.63 % (MSE = 0.36)	7.785 s	64.24 s	99.60 % (MSE = 0.40)

Table 3: Results for multilabel classification on mediamill and delicious datasets. The table shows the training time and classification accuracy using the original label space(denoted by O) and the best results (over 5 runs) using the compressed space(denoted by C). We also report the reconstruction time for OMP. The k parameter and the number of parameters is tuned to obtain the best tradeoff in accuracy and time. Increasing the number of projections further does not lead to vast improvements in accuracy. For the mediamill dataset, we use 25 projections with $k = 5$ whereas for the delicious dataset we use 10 projections with $k = 2$.

We observe that the compressed sensing approach to multilabel classification is faster and results in about 2x speedup considering both the training time in the compressed space and the reconstruction time. Further we observe that the output space for the delicious dataset can be compressed from 983 labels to 10 labels without any considerable loss in classification accuracy. We see that greater number of labels and sparsity in the output space will lead to higher computational savings for the compressed sensing based approach. This also provides an alternate scheme of storing large datasets. We observe that reconstruction time is the bottleneck for the classification pipeline and plan to explore more efficient reconstruction algorithms.

4.3 Weight space

Regularization is a means for enforcing sparsity in the weight space. As explained previously, an L1 penalty added to the loss function helps shrink the weights and encourages only a few weights to have large non-zero values. Thus there are only a few important features which are responsible for the classification. This makes the classification more stable and is helpful for tasks such as feature selection. We reviewed the importance of regularization for classification tasks in section 2. Regularization as mainly a means for avoiding overfitting is well evaluated and is common practice in machine learning. The regularization parameter needs to tradeoff between fitting the data and preventing the weights from becoming too large. It is a hyperparameter and is usually determined by cross-validation. We conduct a simple lasso regression (a squared loss function with an L1 penalty on the weights) experiment to emphasize the importance of regularization. Figure 4 show the out of sample classification accuracy as λ is varied. The figure emphasizes the importance of enforcing sparsity in the weight space.

5 Conclusion and Future Work

In this study, we explored the importance of sparsity for the supervised learning scenario. In particular, we exploited the sparsity in the input space using two sparse coding algorithms. We saw that sparse coding improves the classification performance as well as reduces the time required for training the classifier. We also saw the importance of enforcing sparsity in the weight space using regularization. We used a compressive sensing approach to use the sparsity in the output space for the problem of multilabel classification.

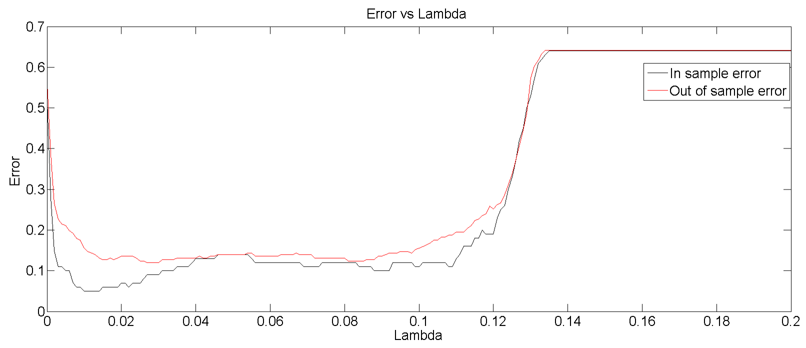


Figure 4: Effect of regularization on in-sample and out-of-sample error. We use the ionosphere dataset from the UCI repository and use lasso for classification. When λ is near zero, the out of sample error is high since we overfit the data. The optimal value of λ is around 0.05 when the out-of-sample error is minimum. As λ increases further, less emphasis is placed on the loss function and leads to underfitting and a high out of sample error.

This reduces the training time for the classifier without sacrificing on the classification performance. We conclude that exploiting sparsity in supervised learning can lead to big gains in performance and time.

We conclude the paper by outlining some promising directions for future work. We plan to explore the ideas presented in [9] which obtains the sparse codes and performs the classification simultaneously. We observe the ADMM based algorithm is slow and does not give robust basis functions as compared to the autoencoder based approach. Agarwal et.al prove some guarantees on the dictionary learnt using ADMM in [2]. They suggest an initialization scheme which provably improves the convergence of the algorithm. We plan to explore this direction in the future. We also plan to combine the sparse coding algorithms with other classifiers other than neural networks [22] and SVMs [32] and quantify the differences in performance and the effect of sparse coding. For multilabel classification, we plan to use more sophisticated reconstruction algorithms both in terms of their reconstruction performance and time. We propose to use structured matrices like circulant or Hadamard matrices to reduce the time for reconstruction.

References

[1] `http://yann.lecun.com/exdb/mnist/`.

[2] Alekh Agarwal, Animashree Anandkumar, Prateek Jain, Praneeth Netrapalli, and Rashish Tandon. Learning sparsely used overcomplete dictionaries via alternating minimization. *arXiv preprint arXiv:1310.7991*, 2013.

[3] Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[4] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *Information Theory, IEEE Transactions on*, 52(12):5406–5425, 2006.

[5] Nicolò Cesa-Bianchi, Claudio Gentile, Luca Zaniboni, et al. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7(1), 2006.

[6] Amanda Clare and Ross D King. Knowledge discovery in multi-label phenotype data. In *Principles of data mining and knowledge discovery*, pages 42–53. Springer, 2001.

[7] David L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[8] George H Dunteman. *Principal components analysis*. Number 69. Sage, 1989.

[9] Alhussein Fawzi, Mike Davies, and Pascal Frossard. Dictionary learning for fast classification based on soft-thresholding. *arXiv preprint arXiv:1402.1973*, 2014.

[10] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[11] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[12] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *NIPS*, volume 22, pages 772–780, 2009.

[13] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. Learning with structured sparsity. *The Journal of Machine Learning Research*, 12:3371–3412, 2011.

[14] Aapo Hyvärinen, Jarmo Hurri, and Patrik O Hoyer. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.*, volume 39. Springer, 2009.

[15] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[16] Kevin Jarrett, Koray Kavukcuoglu, M Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[17] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.

[18] Michael S Lewicki and Terrence J Sejnowski. Learning overcomplete representations. *Neural computation*, 12(2):337–365, 2000.

[19] Bruno A Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[20] Bruno A Olshausen and David J Field. Natural image statistics and efficient coding*. *Network: computation in neural systems*, 7(2):333–339, 1996.

[21] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[22] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2006.

[23] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[24] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[25] Michael P Sceniak, Michael J Hawken, Robert Shapley, et al. Visual spatial characterization of macaque v1 neurons. *Journal of Neurophysiology*, 85(5):1873–1887, 2001.

[26] Cees GM Snoek, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 421–430. ACM, 2006.

[27] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[28] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.

[29] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[30] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD08)*, pages 30–44, 2008.

[31] John Wright, Allen Y Yang, Arvind Ganesh, Shankar S Sastry, and Yi Ma. Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):210–227, 2009.

[32] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1794–1801. IEEE, 2009.

[33] M Zhang and Z Zhou. A review on multi-label learning algorithms. 2013.