The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization

Adam Coates ACOATES@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Stanford University, 353 Serra Mall, Stanford, CA 94305

Abstract

While vector quantization (VQ) has been applied widely to generate features for visual recognition problems, much recent work has focused on more powerful methods. In particular, sparse coding has emerged as a strong alternative to traditional VQ approaches and has been shown to achieve consistently higher performance on benchmark datasets. Both approaches can be split into a training phase, where the system learns a dictionary of basis functions, and an encoding phase, where the dictionary is used to extract features from new inputs. In this work, we investigate the reasons for the success of sparse coding over VQ by decoupling these phases, allowing us to separate out the contributions of training and encoding in a controlled way. Through extensive experiments on CIFAR, NORB and Caltech 101 datasets, we compare several training and encoding schemes, including sparse coding and a form of VQ with a soft threshold activation function. Our results show not only that we can use fast VQ algorithms for training, but that we can just as well use randomly chosen exemplars from the training set. Rather than spend resources on training, we find it is more important to choose a good encoder—which can often be a simple feed forward non-linearity. Our results include state-of-the-art performance on both CIFAR and NORB.

1. Introduction

A great deal of work in computer vision has used vector quantization (VQ) as a tool for constructing higher level image representations. For instance, the K-means algorithm is often used in "visual word models" (Csurka et al., 2004; Lazebnik et al., 2006) to train a dictionary of exemplar low-level descriptors that are then used to define a mapping (an "encoding") of the descriptors into a new feature space. More recently, machine learning research has sought to employ more powerful algorithms and models for these problems that generate better features than those learned with VQ methods. One alternative to VQ that has served in this role is sparse coding, which has consistently yielded better results on benchmark recognition tasks (Yang et al., 2009; Boureau et al., 2010). A natural question is whether this higher performance is the result of learning a better dictionary for representing the structure of the data, or whether sparse codes are simply better non-linear features. In either case, are there other training algorithms or encodings that might be simpler yet competitive with sparse coding? We attempt to answer these questions through a large array of experiments on CIFAR, NORB and Caltech 101 datasets where we carefully separate out the contributions of the training and encoding methods.

More specifically, we note that feature learning algorithms are typically broken into two components: (i) a training algorithm that learns a set of basis functions, D, (referred to variously as "weights", a "codebook", or a "dictionary"), and (ii) an encoding algorithm that, given D, defines a mapping from an input vector x to a feature vector f. Even though these two components are often closely connected, it is not strictly necessary to use an encoding algorithm that matches the training algorithm. For instance, while it is natural to pair K-means training with a hard-assignment encoding scheme, it has been shown that soft encodings (e.g., using Gaussian RBFs) yield better features even when hard assignment was used during training (van Gemert et al., 2008; Boureau et al., 2010; Agarwal & Triggs, 2006). In our experiments, we will exploit the ability to "mix and match" training and encoding algorithms in this way to analyze the contributions of each

Appearing in *Proceedings of the 28^{th} International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

module in a controlled setting. In particular, we will analyze the benefits of sparse coding both as a training algorithm and as an encoding strategy in comparison to several other methods, including VQ.

The main contributions of our work emerge from our analysis of these experiments. We discover two surprising results:

1. When using sparse coding as the encoder, virtually any training algorithm can be used to create a suitable dictionary. We can use VQ, or even randomly chosen exemplars to achieve very high performance.

2. Regardless of the choice of dictionary, a very simple encoder (a soft thresholding function) can often be competitive with sparse coding.

These results not only shed light on the reasons for sparse coding's success (namely, it is a highly effective encoding scheme), but also suggests that we may be able to build and test large models very rapidly, with far simpler training and encoding methods than sparse coding itself.

We begin with an overview of some related work on sparse coding, visual word models, and feature encoding schemes in Section 2 followed by our experimental setup in Section 3. We then continue with our results in Section 4 and conclude with some discussion of our findings and their relationship to prior results in Section 5 before closing.

2. Related Work

Vector quantization has been used extensively in "visual words" models in computer vision. Specifically, the K-means clustering algorithm is used to learn a set of centroids that are then used to map inputs into a new feature space. For instance, in the "bag of words" and spatial pyramid models (Csurka et al., 2004; Lazebnik et al., 2006) it is typical to map an input x to a 1-of-K coded vector s, where the element s_i is 1 if the input vector x belongs to cluster i. This quantization makes K-means learning very fast, but results in crude features. Thus, other authors have used "soft" assignments (e.g., Gaussian activations) to improve performance of the encoding stage (van Gemert et al., 2008; Agarwal & Triggs, 2006).

In our work, we will also use soft assignments with vector quantization, but with some important differences. First, we use a different variant of vector quantization (known as "gain shape" vector quantization) that learns normalized basis functions with dot-products as the similarity metric (rather than Euclidean distance as with K-means). This makes the resulting algorithm more comparable to sparse coding. Second, we use a soft threshold function for our soft assignment, which we will show gives excellent performance and is loosely related to both sparse coding and to recent results using K-means.

The soft threshold function (namely, $\text{sign}(z)\max(0, |z| - \alpha)$ where α is an adjustable threshold) has also been adopted in other recent work. It has been used in conjunction with the Predictive Sparse Decomposition algorithm (Kavukcuoglu et al., 2008), where a feed-forward network is trained explicitly to mimic sparse coding. It has also become popular as the non-linearity in various deep learning architectures (Kavukcuoglu et al., 2010; Nair & Hinton, 2010; Krizhevsky, 2010), and is often referred to as a "shrinkage" function for its role in regularization and sparse coding algorithms (Gregor & LeCun, 2010). Thus, we are by no means the first to observe the usefulness of this particular activation function. In our work, however, we will show that such a nonlinearity on its own is consistently able to compete with sparse coding in our experiments, even without any form of training to "tune" the basis functions to work in conjunction with it.

Though vector quantization is extremely fast, sparse coding has been shown to work consistently better (Boureau et al., 2010; Kavukcuoglu et al., 2010; Yang et al., 2009). Thus, fast algorithms and approximations have been devised to make its use more practical on large problems (Gregor & LeCun, 2010; Wu & Lange, 2008). Other authors, however, have chosen instead to dissect sparse coding in search of its strengths and weaknesses, with an eye toward developing new encodings. In particular, Yu et al. (2009) have argued for "locality preserving" encodings based on the idea that such encodings allow higher-level systems to learn functions across the data manifold more easily. Their system used a locality-aware variant of sparse coding, but the feed-forward encoder we use may also have similar properties while being much simpler.

Finally, we note that recent results in the literature also indicate that the choice of basis functions may not be as critical as one might imagine. Jarrett et al. (2009) showed that architectures with random weights could perform surprisingly well in recognition tasks, even though the performance was not as good as trained weights. Meanwhile, Wang et al. (2010) showed that K-means could be used to learn approximate but similarly performing dictionaries for use with their own encoding. Our results corroborate and extend these findings. In particular, we will provide results using dictionaries created from random noise, randomly sampled exemplars, and vector quantization and show that the last two yield perfectly usable dictionaries in every case.

3. Learning framework

Given an unsupervised learning algorithm, we learn a new feature representation from unlabeled data by employing a common framework. Our feature learning framework is like the patch-based system presented in (Coates et al., 2011) with a few modifications. It shares most of its key components with prior work in visual word models (Csurka et al., 2004; Lazebnik et al., 2006; Agarwal & Triggs, 2006).

In order to generate a set of features, our system first accumulates a batch of small image patches or image descriptors harvested from unlabeled data. When learning from raw pixels, we extract 6 pixel square patches, yielding a bank of vectors that are then normalized[1] and ZCA whitened (Hyvarinen & Oja, 2000) (retaining full variance). If we are learning from SIFT descriptors, we simply take single descriptors to form a bank of 128-dimensional vectors.

Given the batch of input vectors, $x^{(i)} \in \mathbb{R}^n$, an unsupervised learning algorithm is then applied to learn a dictionary of d elements, $D \in \mathbb{R}^{n \times d}$, where each column $D^{(j)}$ is one element. In order to make all of our algorithms consistent (so that we may freely change the choice of encoder), we will make certain that each of the algorithms we use produces normalized dictionary elements: $||D^{(j)}||_2^2 = 1$.

In this work, we will use the following unsupervised learning algorithms for training the dictionary D:

1. **Sparse coding (SC):** We train the dictionary using the L1-penalized sparse coding formulation. That is, we optimize

$$\min_{D,s(i)} \sum_i ||Ds^{(i)} - x^{(i)}||_2^2 + \lambda ||s^{(i)}||_1 \qquad (1)$$
$$\text{subject to } ||D^{(j)}||_2^2 = 1, \forall j$$

   using alternating minimization over the sparse codes, $s^{(i)}$, and the dictionary, D. We use the coordinate descent algorithm to solve for the sparse codes (Wu & Lange, 2008).

2. **Orthogonal matching pursuit (OMP-k):** Similar to sparse coding, the dictionary is trained using an alternating minimization of

$$\min_{D,s(i)} \sum_i ||Ds^{(i)} - x^{(i)}||_2^2 \qquad (2)$$
$$\text{subject to } ||D^{(j)}||_2^2 = 1, \forall j$$
$$\text{and } ||s^{(i)}||_0 \le k, \forall i$$

   where $||s^{(i)}||_0$ is the number of non-zero elements in $s^{(i)}$. In this case, the codes $s^{(i)}$ are computed (approximately) using Orthogonal Matching Pursuit (Pati et al., 1993; Blumensath & Davies, 2007) to compute codes with at most k non-zeros (which we refer to as "OMP-k"). For a single input $x^{(i)}$, OMP-k begins with $s^{(i)} = 0$ and at each iteration greedily selects an element of $s^{(i)}$ to be made non-zero to minimize the residual reconstruction error. After each selection, $s^{(i)}$ is updated to minimize $||Ds^{(i)} - x^{(i)}||_2^2$ over $s^{(i)}$ allowing only the selected elements to be non-zero.

   Importantly, OMP-1 is a form of "gain-shape vector quantization" (and is similar to K-means when the data $x^{(i)}$ and the dictionary elements $D^{(j)}$ all have unit length). Specifically, it chooses $k = \arg\max_j |D^{(j)\top} x^{(i)}|$, then sets $s_k^{(i)} = D^{(k)\top} x^{(i)}$ and all other elements of $s^{(i)}$ to 0. Holding these "one hot" codes fixed, it is then easy to solve for the optimal D in (2).

3. **Sparse RBMs and sparse auto-encoders (RBM, SAE):** In some of our experiments, we train sparse RBMs (Hinton et al., 2006) and sparse auto-encoders (Ranzato et al., 2007; Bengio et al., 2006), both using a logistic sigmoid nonlinearity $g(Wx + b)$. These algorithms yield a set of weights W and biases b. To obtain the dictionary, D, we simply discard the biases and take $D = W^\top$, then normalize the columns of D.

4. **Randomly sampled patches (RP):** We also use a heuristic method for populating the dictionary: we fill the columns of D with normalized vectors sampled randomly from amongst the $x^{(i)}$.

5. **Random weights (R):** It has also been shown that completely random weights can perform surprisingly well in some tasks (Jarrett et al., 2009; Saxe et al., 2010). Thus, we have also tried filling the columns of D with vectors sampled from a unit normal distribution (subsequently normalized to unit length).

After running any of the above training procedures, we have a dictionary D. We must then define an "encoder" that, given D, maps a new input vector x to

[1] We subtract the mean and divide by the standard deviation of the pixel values.

a vector of features, f. We will use the following encoders:

1. **Sparse coding (SC):** Given a dictionary D, which may or may not have been trained using sparse coding, we solve for the sparse code s for x by minimizing (1) with D fixed. Note that the choice of λ in this case may be different from that used during training. We then take:

$$f_j = \max\{0, s\}$$
$$f_{j+d} = \max\{0, -s\}$$

   That is, we split the positive and negative components of the sparse code s into separate features. This allows the higher-level parts of the system (i.e., the classifier) to weight positive and negative responses differently if necessary.[2]

2. **Orthogonal matching pursuit (OMP-k):** As above, we compute s given x and D using OMP-k to yield at most k non-zeros. When $k = 1$, s will have just one non-zero element (equal to $D^{(j)\top}x$, for one choice of j). Given s, the features f are defined as for sparse coding above.

3. **Soft threshold (T):** We use a simple feed-forward non-linearity with a fixed threshold α:

$$f_j = \max\left\{0, D^{(j)\top}x - \alpha\right\}$$
$$f_{j+d} = \max\left\{0, -D^{(j)\top}x - \alpha\right\}$$

4. **"Natural" encoding:** Finally, we will also define the "natural" encoding for a dictionary D as whichever encoding is normally used in conjunction with the training algorithm that generated D. So, for sparse coding with penalty λ, we would use sparse coding with the same penalty, and for OMP we would again use the OMP encoding as above with the same number of non-zeros. For RBMs and auto-encoders we use:

$$f_j = g(W^{(j)}x + b) \tag{3}$$
$$f_{j+d} = g(-W^{(j)}x + b) \tag{4}$$

   where $W^{(j)}$ is the j'th row of W, and g is the logistic sigmoid function.[3] For random patches and random weights we use the soft threshold with $\alpha = 0$ (which corresponds to random linear projections with the positive and negative polarities split into separate features).

Now, given D and a choice of encoder, we have the ability to extract features from a patch or image descriptor x representing a small sub-window of an image. We can then take this feature extractor and sweep it over the entire image to extract a set of feature values to represent the whole image. In our experiments, we use a step (stride) of 1 pixel for CIFAR and NORB (pixel inputs), and a step of 8 pixels for Caltech 101 (SIFT input). The feature values obtained from this extraction process are then pooled (Jarrett et al., 2009; Yang et al., 2009) to form a final feature vector for classification. Depending on the dataset, we use different types of pooling, which we will specify later.

Given a set of labels, we then standardize the data[4] and train a linear classifier (L2-SVM).

Table 1. Cross-validation results for combinations of learning algorithms and encoders on CIFAR-10. All numbers are percent accuracy. The reported accuracies are from 5-fold cross validation on the CIFAR training set, maximizing over the choice of hyper-parameters for both the training and encoding algorithm. I.e., these are the best results we can obtain using the given combination of training and encoding algorithms if we choose the hyper-parameters to maximize the CV accuracy.

Train / Encoder	Natural	SC	OMP-1	OMP-10	T
R	70.5	74.0	65.8	68.6	73.2
RP	76.0	76.6	70.1	71.6	78.1
RBM	74.1	76.7	69.5	72.9	78.3
SAE	74.8	76.5	68.8	71.5	76.7
SC	77.9	78.5	70.8	75.3	78.5
OMP-1	71.4	78.7	71.4	76.0	78.9
OMP-2	73.8	78.5	71.0	75.8	79.0
OMP-5	75.4	78.8	71.0	76.1	79.1
OMP-10	75.3	79.0	70.7	75.3	79.4

4. Experimental Results

4.1. Comparison on CIFAR-10

Our first and most expansive set of experiments are conducted on the CIFAR-10 dataset (Krizhevsky, 2009). Here, we perform a full comparison of all of the learning and encoding algorithms described in Section 3. In particular, we train the dictionary with

[2]This polarity splitting has always improved performance in our experiments, and can be thought of as a more flexible form of the absolute value rectification in (Jarrett et al., 2009), or as non-negative sparse coding with the dictionary $[-D\ D]$.

[3]This "two sided" encoder ensures that we have still get $2d$ features, and do not put the RBM and auto-encoder at a disadvantage relative to the other encodings.

[4]We subtract the mean and divide by the standard deviation of each feature in the training set.

Table 2. Test results for some of the best systems of Table 1 on CIFAR-10. All numbers are percent accuracy.

Train / Encoder	Test Acc.
RP / T	79.1%
SC / SC	78.8%
SC / T	78.9%
OMP-1 / SC	78.8%
OMP-1 / T	79.4%
OMP-10 / T	80.1%
OMP-1 / T ($d = 6000$)	**81.5%**
(Coates et al., 2011) 1600 features	77.9%
(Coates et al., 2011) 4000 features	79.6%
Improved LCC (Yu & Zhang, 2010)	74.5%
Conv. DBN (Krizhevsky, 2010)	78.9%
Deep NN (Ciresan et al., 2011)	80.49%

1600 entries from whitened, 6 by 6 pixel color image patches (108-dimensional vectors), using sparse coding (SC), orthogonal matching pursuit (OMP) with $k = 1, 2, 5, 10$, sparse RBMs (RBM), sparse auto-encoders (SAE), randomly sampled image patches (RP), and random weights (R).

For each dictionary learned with the algorithms above, we then extract features not only using the "natural" encoding associated with the learning algorithm, but also with a host of other encodings from the ones described in Section 3. Specifically, we use sparse coding, with $\lambda \in \{0.5, 0.75, 1.0, 1.25, 1.5\}$, OMP with $k = 1, 10$, and soft thresholding (T) with $\alpha \in \{0.1, 0.25, 0.5, 1.0\}$. After computing the features for a combination of dictionary and encoding method, we construct a final feature vector by average pooling over the 4 image quadrants, yielding $4 \times 2 \times 1600 = 12800$ features. We report the best 5-fold cross-validation results, maximizing over the choice of hyper-parameters[5], for each combination of training algorithm and encoding algorithm in Table 1.

From these numbers, a handful of trends are readily apparent. First, we note that the first column (which pairs each learning algorithm with its standard encoder) shows that sparse coding is superior to all of the other methods by a fairly significant margin, with 77.9% accuracy. OMP-1 (which is just slightly more powerful than hard-assignment K-means) is far worse (71.4%). However, we do get surprisingly close with OMP-10 and random patches. If we look at the results in the remaining columns, it becomes clear that this is not due to the learned basis functions: when using sparse coding as the activation (column 2), *all* of the dictionaries, except the random one, perform competitively. This suggests that the strength of sparse coding on CIFAR comes not from the learned basis functions, but primarily from the encoding mechanism.

[5]Note that for sparse coding this means that the number reported for the "natural" encoding is for the best choice of λ when using the same penalty for both training and encoding. The number in the "sparse coding" column is the best performance possible when choosing *different* λ for training and encoding.

Table 3. Test accuracies for the NORB jittered-cluttered dataset. All numbers are percent accuracy.

Train \ Encoder	Natural	SC ($\lambda = 1$)	T ($\alpha = 0.5$)
R	91.9	93.8	93.1
RP	92.8	**95.0**	93.6
SC $\lambda = 1$	94.1	94.1	93.5
OMP-1	90.9	94.2	92.6
Conv.Net (Scherer et al., 2010)			**94.4%**
SVM-Conv.Net (Huang & LeCun, 2006)			94.1%
ReLU RBM (Nair & Hinton, 2010)			84.8%

Another striking result of these experiments is the success of the soft threshold activation function. Despite using only a feed-forward non-linearity with a *fixed* threshold, this encoding also performs uniformly well across dictionaries, and as well or even better than sparse coding.

We next take several of the best performing systems according to the cross-validation results in Table 1, retrain the classifiers on the full CIFAR training set and then test them on the standard test set. The final test results are reported in Table 2. We note several key numbers. First, using a dictionary of random patches and a soft threshold, we obtain 79.1% accuracy. This is very surprising since this algorithm requires *no training* beyond the choice of the threshold ($\alpha = 0.25$). All of the other results are similar, with just more than 1% separating them. The best overall system identified by cross-validation was OMP-10 with the soft threshold, achieving 80.1% accuracy.

In addition, we note that it is often possible to achieve better performance simply by using much larger dictionaries (van Gemert et al., 2008). This is easily achieved with inexpensive training and encoding algorithms like OMP-1 and the soft-threshold. If we use a dictionary with $d = 6000$ basis vectors, we can achieve 81.5% accuracy—the best known result on CIFAR.

4.2. Experiments on NORB

We also perform experiments on the NORB (jittered-cluttered) dataset (LeCun et al., 2004). Each 108x108 image includes 2 gray stereo channels. We resize the images to 96x96 pixels and average-pool over a 5x5 grid. We train on the first 2 folds of training data (58320 examples), and test on both folds of test data (58320 examples). Based on our experience with CI-

FAR, we chose fixed values for hyper-parameters for these experiments. For sparse coding, we have used $\lambda = 1.0$ and for the soft threshold $\alpha = 0.5$, though the test results are mostly insensitive to these choices.

We report test errors achieved with the natural encoder for each dictionary as well as sparse coding and the soft threshold in Table 3. Again we see that the soft threshold, even when coupled with randomly sampled patches, performs nearly as well as sparse coding. Though performance is slightly lower, we note that its best showing (93.6%) is achieved with far less labor: the sparse coding system requires over 7 hours to run on 40 2.26GHz cores, while the soft threshold scheme requires just 1 hour. In addition, we also see that sparse coding performs comparably regardless of which training algorithm we use. Surprisingly, when using random patches we achieve 95.0% accuracy—better than previously published results for this dataset. For comparison, a recent convolutional neural network system (using max pooling) (Scherer et al., 2010) achieved 94.4% on this dataset.

4.3. Experiments on Caltech 101

Finally, we also performed experiments on the Caltech 101 dataset. For these experiments, we adopted the system of Yang et al. (2009). This system uses SIFT descriptors as the input to feature learning instead of raw pixels. In particular, SIFT descriptors are extracted from each image over a grid. This yields a representation of the image as a set of 128-dimensional vectors, with one descriptor representing each patch of the grid. These vectors become the inputs $x \in \mathbb{R}^{128}$ to the training and encoding algorithms and play the same role as the patches of pixels in our previous experiments.

Given the inputs $x^{(i)}$, a dictionary is constructed as before using random noise (R), randomly sampled descriptors (RP), sparse coding (SC), or vector quantization (OMP-1). After performing the encoding, the features are pooled using max-pooling in a 3-level spatial pyramid (Lazebnik et al., 2006) (i.e., we pool over 4x4, 2x2, and 1x1 grids). We use 30 training examples per class in our experiments, and report the average accuracy over 5 samplings of the training and test sets in Table 4. We use $\lambda = 0.15$ (the same used in (Yang et al., 2009)), and again $\alpha = 0.5$ for the soft threshold.

As can be seen in Table 4 the results are similar, though not identical to those on CIFAR and NORB. First, these results confirm that the choice of dictionary is not especially critical: when using sparse coding as the encoder, we can use randomly sampled descriptors and achieve high performance. However, it appears that the soft threshold works less well for this dataset. It turns out that one shortcoming of the soft threshold activation is the use of a constant threshold. If we instead use a variable threshold (and a dictionary trained with sparse coding), setting α dynamically to yield exactly 20 non-zeros for each example, we achieve 70.1% accuracy ($\pm 0.9\%$). Still a gap remains, which appears to be a result of having few training examples—we will discuss this further in the next section.

Table 4. Test results for the Caltech 101 dataset. Numbers are percent accuracy (and standard deviation) with 30 training images per class.

	SC ($\lambda = 0.15$)	T ($\alpha = 0.5$)
R	67.2% (0.8%)	66.6% (0.2%)
RP	72.6% (0.9%)	64.3% (1.2%)
SC	72.6% (0.9%)	67.7% (0.3%)
OMP-1	71.9% (0.9%)	63.2% (1.4%)
SC-SPM (Yang et al., 2009)		73.2% (0.54%)
(Boureau et al., 2010)		75.7% (1.1%)
(Jarrett et al., 2009)		65.5%

5. Discussion

5.1. Sparse coding and small datasets

A primary distinction between the Caltech 101 dataset and the CIFAR and NORB datasets is the number of available labeled training examples (just 30 per class for Caltech 101). In this situation, regularization and prior knowledge become much more important since we have very few labels for supervised training. It turns out that sparse coding excels in this scenario: it yields a feature vector that works well even when we have very few labels, and even when we use simple algorithms to populate the dictionary.

We have verified this phenomenon on the CIFAR and STL-10[6] (Coates et al., 2011) datasets. We began with a dictionary learned using OMP-1. We then tested the performance of the sparse-coding and soft-threshold encoders when the SVM training procedure is limited to a small number of labeled examples. For CIFAR, the average test performance over 5 folds of labeled data, for various numbers of labeled examples, is plotted in Figure 1. There it can be seen that the performance of sparse coding and the soft-threshold are essentially identical when we use large labeled training sets, but that sparse coding performs much better for smaller numbers of examples. The STL-10 dataset similarly emphasizes smaller labeled datasets (100 examples per fold), though providing additional unlabeled data. On this dataset, the same phenomenon

[6] http://cs.stanford.edu/~acoates/stl10/

is apparent: on 32x32 downsampled images, sparse coding ($\lambda = 1.0$) achieves 59.0% average accuracy ($\pm 0.8\%$), while the soft-threshold ($\alpha = 0.25$) achieves 54.9% ($\pm 0.4\%$). Thus it appears that sparse coding yields a representation that is consistently better when we do not have many labeled examples, though both results are better than those previously reported in (Coates et al., 2011).

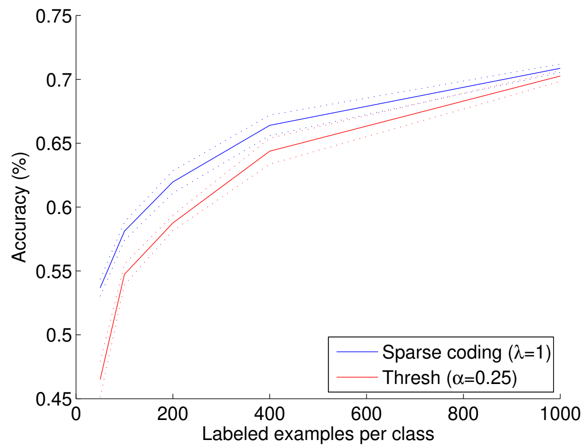

Figure 1. Performance of sparse coding and soft-threshold activations are similar only when we have lots of labeled data as is the case in popular datasets like CIFAR-10.

5.2. Dictionary learning

Our results have shown that the main advantage of sparse coding is as an encoder, and that the choice of basis functions has little effect on performance. Indeed, we can obtain performance on par with any of the learning algorithms tested simply by sampling random patches from the data. This indicates that the main value of the dictionary is to provide a highly overcomplete basis on which to project the data before applying an encoder, but that the exact structure of these basis functions (which comprise the bulk of the parameters that we would normally need to estimate) is less critical than the choice of encoding. All that appears necessary is to choose the basis to roughly tile the space of the input data. This increases the chances that a few basis vectors will be near to an input, yielding a large activation that is useful for identifying the location of the input on the data manifold later (Olshausen & Field, 2004; Yu et al., 2009). This explains why vector quantization is quite capable of competing with more complex algorithms: it simply ensures that there is at least one dictionary entry near any densely populated areas of the input space. We expect that learning is more crucial if we use small dictionaries, since we would then need to be more careful to pick basis functions that span the space of inputs equitably.

6. Conclusion

In this paper we have performed a large array of experiments with several competing forms of unsupervised learning, including sparse coding and vector quantization. By decoupling the training and encoding phases of the algorithms, we have illustrated that the main power of sparse coding is not that it learns better basis functions. In fact, we discovered that any reasonable tiling of the input space (including randomly chosen input patches) is sufficient to obtain high performance on any of the three very different recognition problems that we tested. Instead, the main strength of sparse coding appears to arise from its non-linear encoding scheme, which was almost universally effective in our experiments—even with no training at all. Indeed, it was difficult to beat this encoding on the Caltech 101 dataset. In many cases, however, it was possible to do nearly as well using only a soft threshold function, provided we have sufficient labeled data. Overall, we conclude that most of the performance obtained in our results is a function of the choice of architecture and encoding, suggesting that these are key areas for further study and improvements.

Acknowledgments

This work is supported by the DARPA Deep Learning program under contract number FA8650-10-C-7020. Adam Coates is supported by a Stanford Graduate Fellowship.

References

Agarwal, A. and Triggs, B. Hyperfeatures multilevel local coding for visual recognition. In *European Conference on Computer Vision*, 2006.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems*, 2006.

Blumensath, T. and Davies, M. E. On the difference between orthogonal matching pursuit and orthogonal least squares. Unpublished manuscript, 2007. URL http://www.see.ed.ac.uk/~tblumens/papers/BDOMPvsOLS07.pdf.

Boureau, Y., Bach, F., LeCun, Y., and Ponce, J. Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition*, 2010.

Ciresan, Dan, Meier, Ueli, Masci, Jonathan, Gambardella, Luca Maria, and Schmidhuber, Jürgen. High-performance neural networks for visual object classification. *Pre-print*, 2011. http://arxiv.org/abs/1102.0183.

Coates, Adam, Lee, Honlak, and Ng, Andrew Y. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on AI and Statistics*, 2011.

Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. Visual categorization with bags of keypoints. In *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

Hinton, G.E., Osindero, S., and Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Huang, F.J. and LeCun, Y. Large-scale learning with SVM and convolutional nets for generic object categorization. In *Computer Vision and Pattern Recognition*, 2006.

Hyvarinen, A. and Oja, E. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision*, 2009.

Kavukcuoglu, K., Ranzato, M., and LeCun, Y. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL-TR-2008-12-01, Computational and Biological Learning Lab, Courant Institute, NYU, 2008.

Kavukcuoglu, K., Sermanet, P., Boureau, Y., Gregor, K., Mathieu, M., and LeCun, Y. Learning convolutional feature hierarchies for visual recognition. In *Advances in Neural Information Processing Systems*, 2010.

Krizhevsky, A. Learning multiple layers of features from Tiny Images. Master's thesis, Dept. of Comp. Sci., University of Toronto, 2009.

Krizhevsky, A. Convolutional Deep Belief Networks on CIFAR-10. Unpublished manuscript, 2010.

Lazebnik, S., Schmid, C., and Ponce, J. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition*, 2006.

LeCun, Y., Huang, F., and Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition*, 2004.

Nair, V. and Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on Machine Learning*, 2010.

Olshausen, B.A. and Field, D.J. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4): 481–487, 2004.

Pati, Y. C., Rezaifar, R., and Krishnaprasad, P. S. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Asilomar Conference on Signals, Systems and Computers*, November 1993.

Ranzato, M., Boureau, Y., and LeCun, Y. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems*. 2007.

Saxe, A., Koh, P., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. On random weights and unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Scherer, D., Mller, A., and Behnke, S. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, 2010.

van Gemert, J. C., Geusebroek, J. M., Veenman, C. J., and Smeulders, A. W. M. Kernel codebooks for scene categorization. In *European Conference on Computer Vision*, 2008.

Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., and Gong, Y. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition*, 2010.

Wu, T.T. and Lange, K. Coordinate descent algorithms for lasso penalized regression. *Annals of Applied Statistics*, 2(1), 2008.

Yang, Jianchao, Yu, Kai, Gong, Yihong, and Huang, Thomas S. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition*, 2009.

Yu, K. and Zhang, T. Improved local coordinate coding using local tangents. In *International Conference on Machine Learning*, 2010.

Yu, K., Zhang, T., and Gong, Y. Nonlinear learning using local coordinate coding. In *Advances in Neural Information Processing Systems*, 2009.