

k -Sparse Autoencoders

Alireza Makhzani

Brendan Frey

University of Toronto, 10 King's College Rd. Toronto, Ontario M5S 3G4, Canada

MAKHZANI@PSI.UTORONTO.CA

FREY@PSI.UTORONTO.CA

Abstract

Recently, it has been observed that when representations are learnt in a way that encourages sparsity, improved performance is obtained on classification tasks. These methods involve combinations of activation functions, sampling steps and different kinds of penalties. To investigate the effectiveness of sparsity by itself, we propose the “ k -sparse autoencoder”, which is an autoencoder with linear activation function, where in hidden layers only the k highest activities are kept. When applied to the MNIST and NORB datasets, we find that this method achieves better classification results than denoising autoencoders, networks trained with dropout, and RBMs. k -sparse autoencoders are simple to train and the encoding stage is very fast, making them well-suited to large problem sizes, where conventional sparse coding algorithms cannot be applied.

1. Introduction

Sparse feature learning algorithms range from sparse coding approaches (Olshausen & Field, 1997) to training neural networks with sparsity penalties (Nair & Hinton, 2009; Lee et al., 2007). These methods typically comprise two steps: a learning algorithm that produces a dictionary W that sparsely represents the data $\{\mathbf{x}_i\}_{i=1}^N$, and an encoding algorithm that, given the dictionary, defines a mapping from a new input vector \mathbf{x} to a feature vector.

A practical problem with sparse coding is that both the dictionary learning and the sparse encoding steps are computationally expensive. Dictionaries are usually learnt offline by iteratively recovering sparse codes

and updating the dictionary. Sparse codes are computed using the current dictionary W and a pursuit algorithm to solve

$$\hat{\mathbf{z}}_i = \underset{\mathbf{z}}{\operatorname{argmin}} \|\mathbf{x}_i - W\mathbf{z}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{z}\|_0 < k \quad (1)$$

where \mathbf{z}_i , $i = 1, \dots, N$ are the columns of Z . Convex relaxation methods such as ℓ_1 minimization or greedy methods such as OMP (Tropp & Gilbert, 2007) are used to solve the above optimization. While greedy algorithms are faster, they are still slow in practice. The current sparse codes are then used to update the dictionary, using techniques such as the method of optimal directions (MOD) (Engan et al., 1999) or K-SVD (Aharon et al., 2005). These methods are computationally expensive; MOD requires inverting the data matrix at each step and K-SVD needs to compute a SVD in order to update every column of the dictionary.

To achieve speedups, in (Gregor & LeCun, 2010; Kavukcuoglu et al., 2010), a parameterized non-linear encoder function is trained to explicitly predict sparse codes using a soft thresholding operator. However, they assume that the dictionary is already given and do not address the offline phase.

Another approach that has been taken recently is to train autoencoders in a way that encourages sparsity. However, these methods usually involve combinations of activation functions, sampling steps and different kinds of penalties, and are sometimes not guaranteed to produce sparse representations for each input. For example, in (Lee et al., 2007; Nair & Hinton, 2009), a “lifetime sparsity” penalty function proportional to the negative of the KL divergence between the hidden unit marginals and the target sparsity probability is added to the cost function. This results in sparse activation of hidden units across training points, but does not guarantee that each input has a sparse representation.

The contributions of this paper are as follows. (i) We describe “ k -sparse autoencoders” and show that they can be efficiently learnt and used for sparse coding.

(ii) We explore how different sparsity levels (k) impact representations and classification performance. (iii) We show that by solely relying on sparsity as the regularizer and as the *only nonlinearity*, we can achieve much better results than the other methods, including RBMs, denoising autoencoders (Vincent et al., 2008) and dropout (Hinton et al., 2012). (iv) We demonstrate that k -sparse autoencoders are suitable for pre-training and achieve results comparable to state-of-the-art on MNIST and NORB datasets.

In this paper, Γ is an estimated support set and Γ^c is its complement. W^\dagger is the pseudo-inverse of W and $\text{supp}_k(\mathbf{x})$ is an operator that returns the indices of the k largest coefficients of its input vector. \mathbf{z}_Γ is the vector obtained by restricting the elements of \mathbf{z} to the indices of Γ and W_Γ is the matrix obtained by restricting the columns of W to the indices of Γ .

2. Description of the Algorithm

2.1. The Basic Autoencoder

A shallow autoencoder maps an input vector \mathbf{x} to a hidden representation using the function $\mathbf{z} = f(P\mathbf{x} + \mathbf{b})$, parameterized by $\{P, \mathbf{b}\}$. f is the activation function, e.g., linear, sigmoidal or ReLU. The hidden representation is then mapped linearly to the output using $\hat{\mathbf{x}} = W\mathbf{z} + \mathbf{b}'$. The parameters are optimized to minimize the mean square error of $\|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$ over all training points. Often, tied weights are used, so that $P = W^\top$.

2.2. The k -Sparse Autoencoder

The k -sparse autoencoder is based on an autoencoder with linear activation functions and tied weights. In the feedforward phase, after computing the hidden code $\mathbf{z} = W^\top \mathbf{x} + \mathbf{b}$, rather than reconstructing the input from all of the hidden units, we identify the k largest hidden units and set the others to zero. This can be done by sorting the activities or by using ReLU hidden units with thresholds that are adaptively adjusted until the k largest activities are identified. This results in a vector of activities with the support set of $\text{supp}_k(W^\top \mathbf{x} + \mathbf{b})$. Note that once the k largest activities are selected, the function computed by the network is linear. So the only non-linearity comes from the selection of the k largest activities. This selection step acts as a regularizer that prevents the use of an overly large number of hidden units when reconstructing the input.

Once the weights are trained, the resulting sparse representations may be used for learning to perform downstream classification tasks. However, it has been observed that often, better results are obtained when the

sparse encoding stage used for classification does not exactly match the encoding used for dictionary training (Coates & Ng, 2011). For example, while in k -means, it is natural to have a hard-assignment of the points to the nearest cluster in the encoding stage, it has been shown in (Van Gemert et al., 2008) that soft assignments result in better classification performance. Similarly, for the k -sparse autoencoder, instead of using the k largest elements of $W^\top \mathbf{x} + \mathbf{b}$ as the features, we have observed that slightly better performance is obtained by using the αk largest hidden units where $\alpha \geq 1$ is selected using validation data. So at the test time, we use the support set defined by $\text{supp}_{\alpha k}(W^\top \mathbf{x} + \mathbf{b})$. The algorithm is summarized as follows.

k-Sparse Autoencoders:

Training:

1) Perform the feedforward phase and compute $\mathbf{z} = W^\top \mathbf{x} + \mathbf{b}$

2) Find the k largest activations of \mathbf{z} and set the rest to zero.

$$\mathbf{z}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_k(\mathbf{z})$$

3) Compute the output and the error using the sparsified \mathbf{z} .

$$\hat{\mathbf{x}} = W\mathbf{z} + \mathbf{b}'$$

$$E = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$$

3) Backpropagate the error through the k largest activations defined by Γ and iterate.

Sparse Encoding:

Compute the features $\mathbf{h} = W^\top \mathbf{x} + \mathbf{b}$. Find its αk largest activations and set the rest to zero.

$$\mathbf{h}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_{\alpha k}(\mathbf{h})$$

3. Analysis of the k -Sparse Autoencoder

In this section, we explain how the k -sparse autoencoder can be viewed in the context of sparse coding with incoherent matrices. This perspective sheds light on why the k -sparse autoencoders work and why they achieve invariant features and consequently good classification results. We first explain a sparse recovery algorithm and then show that the k -sparse autoencoder iterates between an approximation of this algorithm and a dictionary update stage.

3.1. Iterative Thresholding with Inversion (ITI)

Iterative hard thresholding (Blumensath & Davies, 2009) is a class of low complexity algorithms, which has recently been proposed for the reconstruction of sparse signals. In this work, we use a variant called “iterative thresholding with inversion” (Maleki, 2009). Given a fixed \mathbf{x} and W , starting from $\mathbf{z}^0 = 0$, ITI iteratively finds the sparsest solution of $\mathbf{x} = W\mathbf{z}$ using the

following steps.

1. **Support Estimation Step:**

$$\Gamma = \text{supp}_k(\mathbf{z}^n + W^\top(\mathbf{x} - W\mathbf{z}^n)) \quad (2)$$

2. **Inversion Step:**

$$\begin{aligned} \mathbf{z}_\Gamma^{n+1} &= W_\Gamma^\dagger \mathbf{x} = (W_\Gamma^\top W_\Gamma)^{-1} W_\Gamma^\top \mathbf{x} \\ \mathbf{z}_{(\Gamma)^c}^{n+1} &= 0 \end{aligned} \quad (3)$$

Assume $H = W^\top W - I$ and \mathbf{z}_0 is the true sparse solution. The first step of ITI estimates the support set as $\Gamma = \text{supp}_k(W^\top \mathbf{x}) = \text{supp}_k(\mathbf{z}_0 + H\mathbf{z}_0)$. If W was orthogonal, we would have $H\mathbf{z}_0 = 0$ and the algorithm would succeed in the first iteration. But if W is overcomplete, $H\mathbf{z}_0$ behaves as a noise vector whose variance decreases after each iteration. After estimating the support set of \mathbf{z} as Γ , we restrict W to the indices included in Γ and form W_Γ . We then use the pseudo-inverse of W_Γ to estimate the non-zero values minimizing $\|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2$. Lastly, we refine the support estimation and repeat the whole process until convergence.

3.2. Sparse Coding with the *k*-Sparse Autoencoder

Here, we show that we can derive the *k*-sparse autoencoder training algorithm by approximating a sparse coding algorithm that uses the ITI algorithm jointly with a dictionary update stage.

The conventional approach of sparse coding is to fix the sparse code matrix Z , while updating the dictionary. However, here, after estimating the support set in the first step of the ITI algorithm, we jointly perform the inversion step of ITI and the dictionary update step, while fixing just the support set of the sparse code Z . In other words, we update the atoms of the dictionary and allow the corresponding non-zero values to change at the same time to minimize $\|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2$ over both W_Γ and \mathbf{z}_Γ .

When we are performing sparse recovery with the ITI algorithm using a fixed dictionary, we should perform a fixed number of iterations to get the perfect reconstruction of the signal. But, in sparse coding, since we learnt a dictionary that is adapted to the signals, as shown in Section 3.3, we can find the support set just with the first iteration of ITI:

$$\Gamma_z = \text{supp}_k(W^\top \mathbf{x}) \quad (4)$$

In the inversion step of the ITI algorithm, once we estimate the support set, we use the pseudo-inverse of

W_Γ to find the non-zero values of the support set. The pseudo-inverse of the matrix W_Γ is a matrix, such as P_Γ , that minimizes the following cost function.

$$\begin{aligned} W_\Gamma^\dagger &= \arg \min_{P_\Gamma} \|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2 \\ &= \arg \min_{P_\Gamma} \|\mathbf{x} - W_\Gamma P_\Gamma \mathbf{x}\|_2^2 \end{aligned} \quad (5)$$

Finding the exact pseudo-inverse of W_Γ is computationally expensive, so instead, we perform a single step of gradient descent. The gradient with respect to P_Γ is found as follows:

$$\frac{\partial \|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2}{\partial P_\Gamma} = \frac{\partial \|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2}{\partial \mathbf{z}_\Gamma} \mathbf{x} \quad (6)$$

The first term of the right hand side of the Equation (6) is the dictionary update stage, which is computed as follows:

$$\frac{\partial \|\mathbf{x} - W_\Gamma \mathbf{z}_\Gamma\|_2^2}{\partial \mathbf{z}_\Gamma} = (W_\Gamma \mathbf{z}_\Gamma - \mathbf{x}) \mathbf{z}_\Gamma^\top \quad (7)$$

Therefore, in order to approximate the pseudo-inverse, we first find the dictionary derivative and then “back-propagate” it to find the update of the pseudo-inverse.

We can view these operations in the context of an autoencoder with linear activations where P is the encoder weight matrix and W is the decoder weight matrix. At each iteration, instead of back-propagating through all the hidden units, we just back-propagate through the units with the *k* largest activities, defined by $\text{supp}_k(W^\top \mathbf{x})$, which is the first iteration of ITI. Keeping the *k* largest hidden activities and ignoring the others is the same as forming W_Γ by restricting W to the estimated support set. Back-propagation on the decoder weights is the same as gradient descent on the dictionary and back-propagation on the encoder weights is the same as approximating the pseudo-inverse of the corresponding W_Γ .

We can perform support estimation in the feedforward phase by assuming $P = W^\top$ (i.e., the autoencoder has tied weights). In this case, support estimation can be done by computing $\mathbf{z} = W^\top \mathbf{x} + \mathbf{b}$ and picking the *k* largest activations; the bias just accounts for the mean and subtracts its contribution. Then the “inversion” and “dictionary update” steps are done at the same time by back-propagation through just the units with the *k* largest activities.

In summary, we can view *k*-sparse autoencoders as the approximation of a sparse coding algorithm which uses ITI in the sparse recovery stage.

3.3. Importance of Incoherence

The coherence of a dictionary indicates the degree of similarity between different atoms or different collections of atoms. Since the dictionary is overcomplete, we can represent each column of the dictionary as a linear combination of other columns. But what incoherence implies is that we should not be able to represent a column as a *sparse* linear combination of other columns and the coefficients of the linear combination should be dense. For example, if two columns are exactly the same, then the dictionary is highly coherent since we can represent one of those columns as the sparse linear combination of the rest of the columns. A naive measure of coherence that has been proposed in the literature is the mutual coherence $\mu(W)$ which is defined as the maximum absolute inner product across all the possible pairs of the atoms of the dictionary.

$$\mu(W) = \max_{i \neq j} |\langle \mathbf{w}_i, \mathbf{w}_j \rangle| \quad (8)$$

There is a close relationship between the coherency of the dictionary and the uniqueness of the sparse solution of $\mathbf{x} = W\mathbf{z}$. In (Donoho & Elad, 2003), it has been proven that if $k \leq (1 + \mu^{-1})$, then the sparsest solution is unique.

We can show that if the dictionary is incoherent enough, there is going to be an attraction ball around the signal \mathbf{x} and there is only one unique sparse linear combination of the columns that can get into this attraction ball. So even if we perturb the input with a small amount of noise, translation, rotation, *etc.*, we can still achieve perfect reconstruction of the original signal and the sparse features are always roughly conserved. Therefore, incoherency of the dictionary is a measure of invariance and stability of the features. This is related to the denoising autoencoder (Vincent et al., 2008) in which we achieve invariant features by trying to reconstruct the original signal from its noisy versions.

Here we show that if the dictionary is incoherent enough, the first step of the ITI algorithm is sufficient for perfect sparse recovery.

Theorem 3.1. Assume $\mathbf{x} = W\mathbf{z}$ and the columns of the dictionary have unit ℓ_2 -norm. Also without loss of generality, assume that the non-zero elements of \mathbf{z} are its first k elements and are sorted as $z_1 \geq z_2 \geq \dots \geq z_k$. Then, if $k\mu \leq \frac{z_k}{2z_1}$, we can recover the support set of \mathbf{z} using $\text{supp}_k(W^\top \mathbf{x})$.

Proof: Let us assume $0 \leq i \leq k$ and $\mathbf{y} = W^\top \mathbf{x}$. Then, we can write:

$$y_i = z_i + \sum_{j=1, j \neq i}^k \langle \mathbf{w}_i, \mathbf{w}_j \rangle z_j \geq z_i - \mu \sum_{j=1, j \neq i}^k z_j \geq z_k - k\mu z_1 \quad (9)$$

On the other hand:

$$\max_{i > k} \{y_i\} = \max_{i > k} \left\{ \sum_{j=1}^k \langle \mathbf{w}_i, \mathbf{w}_j \rangle z_j \right\} \leq k\mu z_1 \quad (10)$$

So if $k\mu \leq \frac{z_k}{2z_1}$, all the first k elements of \mathbf{y} are guaranteed to be greater than the rest of its elements.

As we can see from Theorem 3.1, the chances of finding the true support set with the encoder part of the k -sparse autoencoder depends on the incoherency of the learnt dictionary. As the k -sparse autoencoder converges (i.e., the reconstruction error goes to zero), the algorithm learns a dictionary that satisfies $\mathbf{x} \approx W\mathbf{z}$, so the support set of \mathbf{z} can be estimated using the first step of ITI. Since $\text{supp}_k(W^\top \mathbf{x})$ succeeds in finding the support set when the algorithm converges, the learnt dictionary must be sufficiently incoherent.

4. Experiments

In this section, we evaluate the performance of k -sparse autoencoders in both unsupervised learning and in shallow and deep discriminative learning tasks.

4.1. Datasets

We use the MNIST handwritten digit dataset, which consists of 60,000 training images and 10,000 test images. We randomly separate the training set into 50,000 training cases and 10,000 cases for validation.

We also use the small NORB normalized-uniform dataset (LeCun et al., 2004), which contains 24,300 training examples and 24,300 test examples. This database contains images of 50 toys from 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. Each image consists of two channels, each of size 96×96 pixels. We take the inner 64×64 pixels of each channel and resize it using bicubic interpolation to the size of 32×32 pixels from which we form a vector with 2048 dimensions as the input. Data points are subtracted by the mean and divided by the standard deviation along each input dimension across the whole training set to normalize the contrast. The training set is separated into 20,000 for training and 4,300 for validation.

We also test our method on natural image patches extracted from CIFAR-10 dataset. We randomly extract 1000000 patches of size 8×8 from the 50000 32×32 im-

ages of CIFAR-10. Each patch is then locally contrast-normalized and ZCA whitened. This preprocessing pipeline is the same as the one used in (Coates et al., 2011) for feature extraction.

4.2. Training of *k*-Sparse Autoencoders

4.2.1. SCHEDULING OF THE SPARSITY LEVEL

When we are enforcing low sparsity levels in *k*-sparse autoencoders (e.g., $k=15$ on MNIST), one issue that might arise is that in the first few epochs, the algorithm greedily assigns individual hidden units to groups of training cases, in a manner similar to *k*-means clustering. In subsequent epochs, these hidden units will be picked and re-enforced and other hidden units will not be adjusted. That is, too much sparsity can prevent gradient back-propagation from adjusting the weights of these other ‘dead’ hidden units. We can address this problem by scheduling the sparsity level over epochs as follows.

Suppose we are aiming for a sparsity level of $k = 15$. Then, we start off with a large sparsity level (e.g. $k = 100$) for which the *k*-sparse autoencoder can train all the hidden units. We then linearly decrease the sparsity level from $k = 100$ to $k = 15$ over the first half of the epochs. This initializes the autoencoder in a good regime, for which all of the hidden units have a significant chance of being picked. Then, we keep $k = 15$ for the second half of the epochs. With this scheduling, we can train all of the filters, even for low sparsity levels.

4.2.2. TRAINING HYPER-PARAMETERS

We optimized the model parameters using stochastic gradient descent with momentum as follows.

$$\begin{aligned} \mathbf{v}_{k+1} &= m_k \mathbf{v}_k - \eta_k \nabla f(\mathbf{x}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{v}_k \end{aligned} \quad (11)$$

Here, \mathbf{v}_k is the velocity vector, m_k is the momentum and η_k is the learning rate at the k -th iteration. We also use a Gaussian distribution with a standard deviation of σ for initialization of the weights. We use different momentum values, learning rates and initializations based on the task and the dataset, and validation is used to select hyperparameters. In the unsupervised MNIST task, the values were $\sigma = 0.01$, $m_k = 0.9$ and $\eta_k = 0.01$, for 5000 epochs. In the supervised MNIST task, training started with $m_k = 0.25$ and $\eta_k = 1$, and then the learning rate was linearly decreased to 0.001 over 200 epochs. In the unsupervised NORB task, the values were $\sigma = 0.01$, $m_k = 0.9$ and $\eta_k = 0.0001$, for 5000 epochs. In the supervised NORB task, training

started with $m_k = 0.9$ and $\eta_k = 0.01$, and then the learning rate was linearly decreased to 0.001 over 200 epochs.

4.2.3. IMPLEMENTATIONS

While most of the conventional sparse coding algorithms require complex matrix operations such as matrix inversion or SVD decomposition, the *k*-sparse autoencoders only need matrix multiplications and sorting operations in both dictionary learning stage and the sparse encoding stage. (For a parallel, distributed implementation, the sorting operation can be replaced by a method that recursively applies a threshold until k values remain.) We used an efficient GPU implementation obtained using the publicly available *gnumpy* library (Tieleman, 2010) on a single Nvidia GTX 680 GPU.

4.3. Effect of Sparsity Level

In *k*-sparse autoencoders, we are able to tune the value of k to obtain the desirable sparsity level which makes the algorithm suitable for a wide variety of datasets. For example, one application could be pre-training a shallow or deep discriminative neural network. For large values of k (e.g., $k = 100$ on MNIST), the algorithm tends to learn very local features as is shown in Figure 1a and 2a. These features are too primitive to be used for classification using a shallow architecture since a naive linear classifier does not have enough capacity to combine these features and achieve a good classification rate. However, these features could be used for pre-training deep neural nets.

As we decrease the the sparsity level (e.g., $k = 40$ on MNIST), the output is reconstructed using a smaller number of hidden units and thus the features tend to be more global, as can be seen in Figure 1b,1c and 2b. For example, in the MNIST dataset, the lengths of the strokes increase when the sparsity level is decreased. These less local features are suitable for classification using a shallow architecture. Nevertheless, forcing too much sparsity (e.g., $k = 10$ on MNIST), results in features that are too global and do not factor the input into parts, as depicted Figure 1d and 2c.

Fig. 3 shows the visualization of filters of the *k*-sparse autoencoder with 1000 hidden units and sparsity level of $k = 50$ learnt from random image patches extracted from CIFAR-10 dataset. We can see that the *k*-sparse autoencoder has learnt localized Gabor filters from natural image patches.

Fig. 4 plots histograms of the hidden unit activities for various unsupervised learning algorithms, includ-

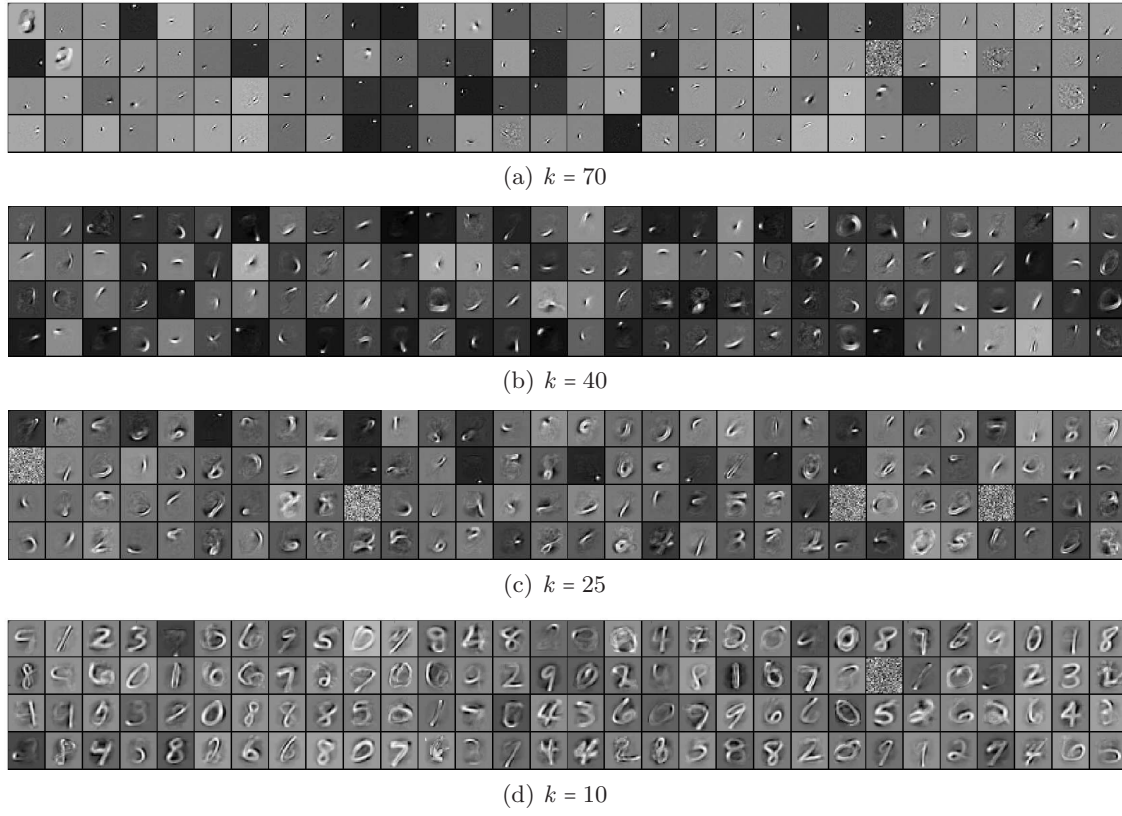


Figure 1. Filters of the k -sparse autoencoder for different sparsity levels k , learnt from MNIST with 1000 hidden units.

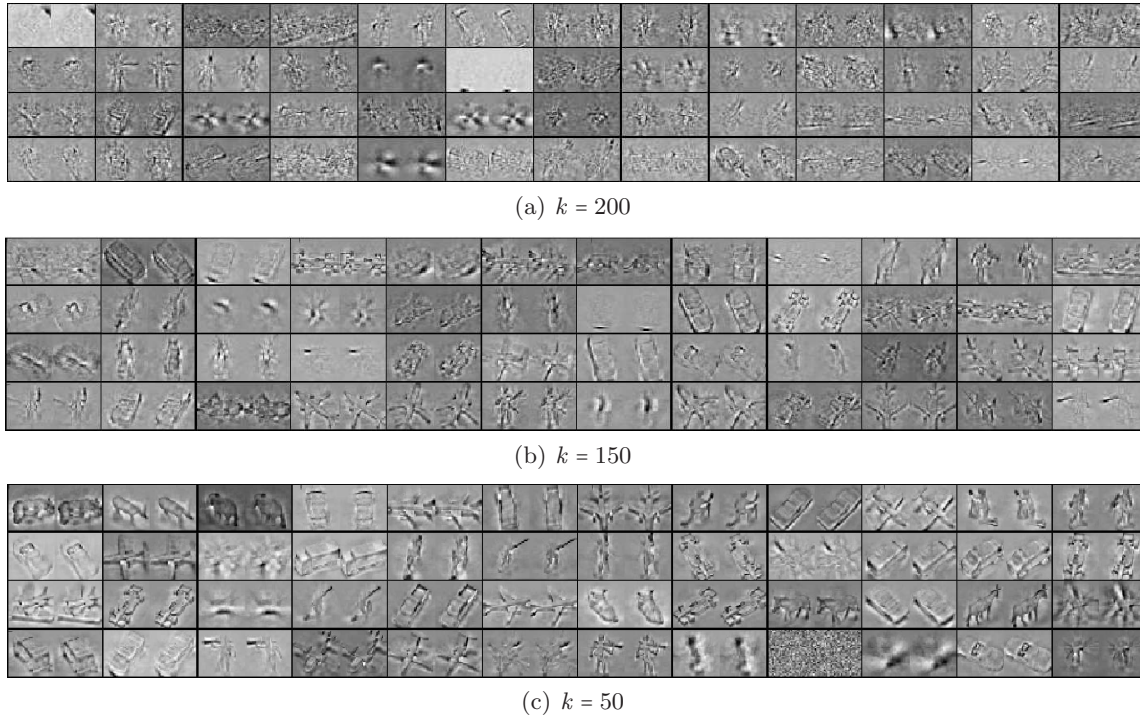


Figure 2. Filters of the k -sparse autoencoder for different sparsity levels k , learnt from NORB with 4000 hidden units.

	Error Rate
Raw Pixels	7.20%
RBM	1.81%
Dropout Autoencoder (50% hidden)	1.80%
Denoising Autoencoder (20% input dropout)	1.95%
Dropout + Denoising Autoencoder (20% input and 50% hidden)	1.60%
<i>k</i> -Sparse Autoencoder, $k = 40$	1.54%
<i>k</i> -Sparse Autoencoder, $k = 25$	1.35%
<i>k</i> -Sparse Autoencoder, $k = 10$	2.10%

Table 1. Performance of unsupervised learning methods (without fine-tuning) with 1000 hidden units on **MNIST**.

ing the *k*-sparse autoencoder ($k=70$ and $k=15$), applied to the MNIST data. This figure contrasts the sparsity achieved by the *k*-sparse autoencoder with that of other algorithms.

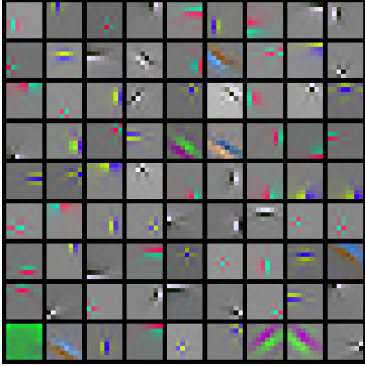


Figure 3. Filters of *k*-sparse autoencoder with 1000 hidden units and $k = 50$, learnt from CIFAR-10 random patches.

4.4. Unsupervised Feature Learning Results

In order to compare the quality of the features learnt by our algorithm with those learnt by other unsupervised learning methods, we first extracted features using each unsupervised learning algorithm. Then we fixed the features and trained a logistic regression classifier using those features. The usefulness of the features is then evaluated by examining the error rate of the classifier.

We trained a number of architectures on the MNIST and NORB datasets, including RBM, dropout autoencoder and denoising autoencoder. In dropout, after finding the features using dropout regularization with a dropout rate of 50%, we used all of the hidden units as the features (this worked best). For the denoising autoencoder, after training the network by dropping the input pixels with a rate of 20%, we used

	Error Rate
Raw Pixels	23%
RBM (weight decay)	10.6%
Dropout Autoencoder	10.1%
Denoising Autoencoder (20% input dropout)	9.5%
<i>k</i> -Sparse Autoencoder, $k = 200$	10.4%
<i>k</i> -Sparse Autoencoder, $k = 150$	8.6%
<i>k</i> -Sparse Autoencoder, $k = 75$	9.5%

Table 2. Performance of unsupervised learning methods (without fine-tuning) with 4000 hidden units on **NORB**.

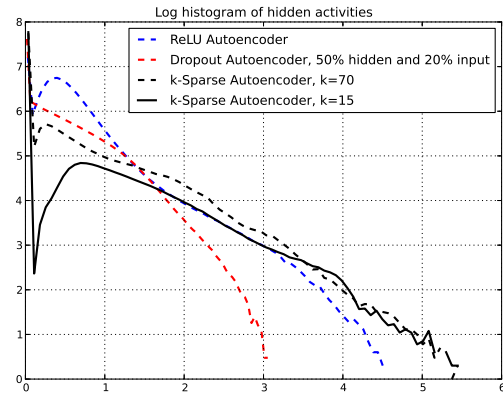


Figure 4. Histogram of hidden unit activities for various unsupervised learning methods.

all of the uncorrupted input pixels to find the features for classification (this worked best). In the *k*-sparse autoencoder, after training the dictionary, we used $\mathbf{h} = \text{supp}_{\alpha k}(W^T \mathbf{x} + \mathbf{b})$ to find the features as explained in Section 2.2, where α was determined using validation data. Results for different architectures are compared in Tables 1, 2. We can see that the performance of our *k*-sparse autoencoder is better than the rest of the algorithms. In our algorithm, the best result is achieved by $k = 25, \alpha = 3$ with 1000 hidden units on MNIST dataset and by $k = 150, \alpha = 2$ with 4000 hidden units on NORB dataset.

4.5. Shallow Supervised Learning Results

In supervised learning, it is a common practice to use the encoder weights learnt by an unsupervised learning method to initialize the early layers of a multilayer discriminative model (Erhan et al., 2010). The back-propagation algorithm is then used

	Error
Without Pre-Training	1.60%
RBM + F.T.	1.24%
Shallow Dropout AE + F.T. (%50 hidden)	1.05%
Denoising AE + F.T. (%20 input dropout)	1.20%
Deep Dropout AE + F.T. (Layer-wise pre-training, %50 hidden)	0.85%
<i>k</i> -Sparse AE + F.T. (<i>k</i> =25)	1.08%
Deep <i>k</i> -Sparse AE + F.T. (Layer-wise pre-training)	0.97%

Table 3. Performance of supervised learning methods on **MNIST**. Pre-training was performed using the corresponding unsupervised learning algorithm with 1000 hidden units, and then the model was fine-tuned.

to adjust the weights of the last hidden layer and also to fine-tune the weights in the previous layers. This procedure is often referred to as discriminative fine-tuning. In this section, we report results using unsupervised learning algorithms such as RBMs, DBNs (Salakhutdinov & Larochelle, 2010), DBMs (Salakhutdinov & Larochelle, 2010), third-order RBM (Nair & Hinton, 2009), dropout autoencoders, denoising autoencoders and *k*-sparse autoencoders to initialize a shallow discriminative neural network for the MNIST and NORB datasets. We used back-propagation to fine-tune the weights. The regularization method used in the fine-tuning stage of different algorithms is the same as the one used in the training of the corresponding unsupervised learning task. For instance, we fine-tuned the weights obtained from dropout autoencoder with dropout regularization or in denoising autoencoder, we fine-tuned the discriminative neural net by adding noise to the input. In a similar manner, in the fine-tuning stage of the *k*-sparse autoencoder, we used the αk largest hidden units in the corresponding discriminative neural network, as explained in Section 2.2. Tables 3 and 4 reports the error rates obtained by different methods.

4.6. Deep Supervised Learning Results

The *k*-sparse autoencoder can be used as a building block of a deep neural network, using greedy layer-wise pre-training (Bengio et al., 2007). We first train a shallow *k*-sparse autoencoder and obtain the hidden codes. We then fix the features and train another *k*-

	Error
Without Pre-Training	12.7%
DBN	8.3%
DBM	7.2%
third-order RBM	6.5%
Shallow Dropout AE + F.T. (%50 hidden)	8.2%
Shallow Denoising AE + F.T. (%20 input dropout)	7.9%
Deep Dropout AE + F.T. (Layer-wise pre-training, %50 hidden)	7.0%
Shallow <i>k</i> -Sparse AE + F.T. (<i>k</i> =150)	7.8%
Deep <i>k</i> -Sparse AE + F.T. (<i>k</i> =150, Layer-wise pre-training)	7.4%

Table 4. Performance of supervised learning methods on **NORB**. Pre-training was performed using the corresponding unsupervised learning algorithm with 4000 hidden units, and then the model was fine-tuned.

sparse autoencoder on top of them to obtain another set of hidden codes. Then we use the parameters of these autoencoders to initialize a discriminative neural network with two hidden layers.

In the fine-tuning stage of the deep neural net, we first fix the parameters of the first and second layers and train a softmax classifier on top of the second layer. We then hold the weights of the first layer fixed and train the second layer and softmax jointly using the initialization of the softmax that we found in the previous step. Finally, we jointly fine-tune all of the layers with the previous initialization. We have observed that this method of layer-wise fine-tuning can improve the classification performance compared to the case where we fine-tune all the layers at the same time.

In all of the fine-tuning steps, we keep the αk largest hidden codes, where $k = 25, \alpha = 3$ in MNIST and $k = 150, \alpha = 2$ in NORB in both hidden layers. Tables 3 and 4 report the classification results of different deep supervised learning methods.

5. Conclusion

In this work, we proposed a very fast sparse coding method called *k*-sparse autoencoder, which achieves exact sparsity in the hidden representation. The main message of this paper is that we can use the resulting representations to achieve state-of-the-art classification results, solely by enforcing sparsity in the hidden units and without using any other nonlinearity or regularization. We also discussed how the *k*-sparse autoencoder could be used for pre-training shallow and

deep supervised architectures.

6. Acknowledgment

We would like to thank Andrew Delong, Babak Alipanahi and Lei Jimmy Ba for the valuable comments.

References

- Aharon, Michal, Elad, Michael, and Bruckstein, Alfred. K-svd: Design of dictionaries for sparse representation. *Proceedings of SPARS*, 5:9–12, 2005.
- Bengio, Yoshua, Lamblin, Pascal, Popovici, Dan, and Larochelle, Hugo. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.
- Blumensath, Thomas and Davies, Mike E. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.
- Coates, Adam and Ng, Andrew. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 921–928, 2011.
- Coates, Adam, Ng, Andrew Y, and Lee, Honglak. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 215–223, 2011.
- Donoho, David L and Elad, Michael. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.
- Engan, Kjersti, Aase, Sven Ole, and Hakon Husoy, J. Method of optimal directions for frame design. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 5, pp. 2443–2446. IEEE, 1999.
- Erhan, Dumitru, Bengio, Yoshua, Courville, Aaron, Manzagol, Pierre-Antoine, Vincent, Pascal, and Bengio, Samy. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.
- Gregor, Karol and LeCun, Yann. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 399–406, 2010.
- Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- Kavukcuoglu, Koray, Ranzato, Marc’Aurelio, and LeCun, Yann. Fast inference in sparse coding algorithms with applications to object recognition. *arXiv preprint arXiv:1010.3467*, 2010.
- LeCun, Yann, Huang, Fu Jie, and Bottou, Leon. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, CVPR*, volume 2, pp. II–97. IEEE, 2004.
- Lee, Honglak, Ekanadham, Chaitanya, and Ng, Andrew. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pp. 873–880, 2007.
- Maleki, Arian. Coherence analysis of iterative thresholding algorithms. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pp. 236–243. IEEE, 2009.
- Nair, Vinod and Hinton, Geoffrey E. 3d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems*, pp. 1339–1347, 2009.
- Olshausen, Bruno A and Field, David J. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.
- Salakhutdinov, Ruslan and Larochelle, Hugo. Efficient learning of deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 693–700, 2010.
- Tieleman, Tijmen. Gnumpy: an easy way to use gpu boards in python. *Department of Computer Science, University of Toronto*, 2010.
- Tropp, Joel A and Gilbert, Anna C. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.
- Van Gemert, Jan C, Geusebroek, Jan-Mark, Veenman, Cor J, and Smeulders, Arnold WM. Kernel codebooks for scene categorization. In *Computer Vision—ECCV 2008*, pp. 696–709. Springer, 2008.
- Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.