Sparse Coding with Multi-layer Decoders using Variance Regularization

Anonymous authors
Paper under double-blind review

Abstract

Sparse representations of images are useful in many computer vision applications. Sparse coding with an l_1 penalty and a learned linear dictionary requires regularization of the dictionary to prevent a collapse in the l_1 norms of the codes. Typically, this regularization entails bounding the Euclidean norms of the dictionary's elements. In this work, we propose a novel sparse coding protocol which prevents a collapse in the codes without the need to regularize the decoder. Our method regularizes the codes directly so that each latent code component has variance greater than a fixed threshold over a set of sparse representations for a given set of inputs. Furthermore, we explore ways to effectively train sparse coding systems with multi-layer decoders since they can model more complex relationships than linear dictionaries. In our experiments with MNIST and natural image patches, we show that decoders learned with our approach have interpretable features both in the linear and multi-layer case. Moreover, we show that sparse autoencoders with multi-layer decoders trained using our variance regularization method produce higher quality reconstructions with sparser representations when compared to autoencoders with linear dictionaries. Additionally, sparse representations obtained with our variance regularization approach are useful in the downstream tasks of denoising and classification in the low-data regime.

1 Introduction

Finding representations of data which are useful for downstream applications is at the core of machine learning and deep learning research (Bengio et al., 2013). Self-supervised learning (SSL) is a branch of representation learning which is entirely data driven and in which the representations are learned without the need for external supervision such as semantic annotations. In SSL, learning can happen through various pre-text tasks such as reconstructing the input with autoencoders (Vincent et al., 2008), predicting the correct order of frames in a video (Misra et al., 2016; Fernando et al., 2017), or enforcing equivariant representations across transformations of the same image (He et al., 2019; Grill et al., 2020; Zbontar et al., 2021; Bardes et al., 2021). The pre-text tasks should be designed to extract data representations useful in other downstream tasks.

We focus on learning representations of images through *sparse coding*, a self-supervised learning paradigm based on input reconstruction (Olshausen & Field, 1997). A representation (also referred to as a code) in the form of a feature vector $\boldsymbol{z} \in \mathbb{R}^l$ is considered sparse if only a small number of its components are active for a given data sample. Sparsity introduces the inductive bias that only a small number of factors are relevant for a single data point and it is a desirable property for data representations as it can improve efficiency and interpretability of the representations (Bengio, 2009; Bengio et al., 2013). Sparse representations of images have also been shown to improve classification performance (Ranzato et al., 2007; Kavukcuoglu et al., 2010; Makhzani & Frey, 2013) and are popular for denoising and inpainting applications (Elad & Aharon, 2006; Mairal et al., 2007). They are also biologically motivated as the primary visual cortex uses sparse activations of neurons to represent natural scenes (Olshausen & Field, 1997; Yoshida & Ohki, 2020).

In sparse coding, the goal is to find a latent sparse representation $\boldsymbol{z} \in \mathbb{R}^l$ which can reconstruct an input $\boldsymbol{y} \in \mathbb{R}^d$ using a *learned decoder* $\mathcal{D}$. Typically, the decoder is linear, namely $\mathcal{D} \in \mathbb{R}^{d \times l}$, and the sparse code

$\boldsymbol{z}$ selects a linear combination of a few of its columns to reconstruct the input. In practice, a sparse code $\boldsymbol{z}$ is computed using an *inference algorithm* which minimizes $\boldsymbol{z}$'s l_1 norm and simultaneously optimizes for a good reconstruction of the input $\boldsymbol{y}$ using $\mathcal{D}$'s columns. Learning of the decoder $\mathcal{D}$ typically happens in an iterative EM-like fashion. Inference provides a set of sparse codes corresponding to some inputs (expectation step). Given these codes, the decoder's parameters are updated using gradient descent to minimize the error between the original inputs and their reconstructions (maximization step). Sparse coding systems in which the decoder is non-linear are hard to learn and often require layer-wise training. In this work, we explore ways to effectively train sparse coding systems with non-linear decoders using end-to-end learning.

In order to successfully train a sparse coding system which uses a linear dictionary $\mathcal{D}$, it is necessary to limit the norms of the dictionary's weights. Indeed, consider a reconstruction $\tilde{\boldsymbol{y}}$ obtained from a dictionary $\mathcal{D}$ and code $\boldsymbol{z}$, namely $\tilde{\boldsymbol{y}} = \mathcal{D}\boldsymbol{z}$. The same reconstruction can be obtained from a re-scaled dictionary $\mathcal{D}' = c\mathcal{D}$ and code $\boldsymbol{z}' = \boldsymbol{z}/c$ for any non-zero multiple c. When $c > 1$, the representation $\boldsymbol{z}'$ has a smaller l_1 norm than $\boldsymbol{z}$ but produces the same reconstruction. In practice, if $\mathcal{D}$'s weights are unbounded during training, they become arbitrarily large, while the l_1 norms of the latent representations become arbitrarily small which leads to a collapse in their l_1 norm. Typically, in order to restrict the dictionary $\mathcal{D}$, its columns are re-scaled to have a constant l_2 norm. However, it is not trivial to extend this normalization procedure to the case when the decoder $\mathcal{D}$ is a multi-layer neural network and at the same time learn useful features in $\mathcal{D}$.

Instead of l_2-normalizing $\mathcal{D}$'s columns, we propose to prevent collapse in the l_1 norm of the codes by applying variance regularization to each latent code component, a strategy similar to the variance principle presented in Bardes et al. (2021). In particular, we add a regularization term to the energy minimized during inference for computing a set of codes. This regularization term encourages the variance of each latent code component to be greater than a fixed lower bound. This strategy ensures that the norms of the sparse codes do not collapse and we find that it removes the need to directly restrict $\mathcal{D}$'s parameters. Since this variance regularization term is independent from $\mathcal{D}$'s architecture, we can use it when the decoder is a deep neural network. In this work, we experiment with fully connected sparse autoencoders in which the decoder has one hidden layer and show that we can train such models successfully using our variance regularization strategy.

The main contributions of our work are:

- We introduce an effective way to train sparse autoencoders and prevent collapse in the l_1 norms of the codes that they produce without the need to regularize the decoder's weights but instead by encouraging the latent code components to have variance above a fixed threshold.

- We show empirically that linear decoders trained with this variance regularization strategy are comparable to the ones trained using the standard l_2-normalization approach in terms of the features that they learn and the quality of the reconstructions they produce. Moreover, variance regularization allows us to successfully extend sparse coding to the case of a non-linear fully-connected decoder with one hidden layer.

- Additionally, our experiments show that sparse representations obtained from autoencoders with a non-linear decoder trained using our proposed variance regularization strategy: 1) produce higher quality reconstructions than ones obtained from autoencoders with a linear decoder, 2) are helpful in the downstream task of classification in the low-data regime.

2 Method

A schematic view of our sparse autoencoder setup is presented in Figure 1. At a high level, given an input $\boldsymbol{y}$ and a fixed decoder $\mathcal{D}$, we perform inference with the FISTA algorithm (Beck & Teboulle, 2009) to find a sparse code $\boldsymbol{z}^*$ which can best reconstruct $\boldsymbol{y}$ using $\mathcal{D}$'s elements. The decoder's weights are trained by minimizing the mean squared error (MSE) between the input $\boldsymbol{y}$ and the reconstruction $\tilde{\boldsymbol{y}}$ computed from $\boldsymbol{z}^*$. The encoder $\mathcal{E}$ is trained to predict FISTA's output $\boldsymbol{z}^*$. More details on each of the components in Figure 1 are presented in the following sections.

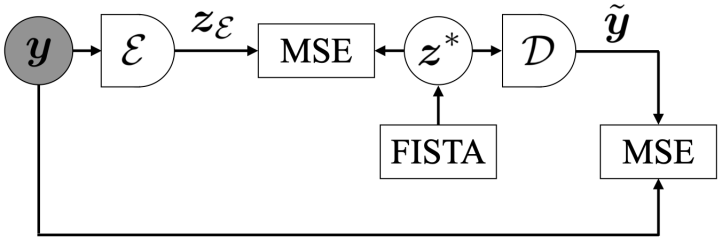


Figure 1: Our sparse autoencoder setup. The encoder $\mathcal{E}$ is trained to predict the codes $\boldsymbol{z}^*$ computed from inference with FISTA. The decoder $\mathcal{D}$ learns to reconstruct inputs $\boldsymbol{y}$ from the sparse codes $\boldsymbol{z}^*$.

2.1 Background: Sparse Coding and Dictionary Learning

Inference Sparse coding algorithms with an l_1 sparsity penalty and a linear decoder $\mathcal{D} \in \mathbb{R}^{d \times l}$ perform inference to find a latent sparse representation $\boldsymbol{z} \in \mathbb{R}^l$ of a given data sample $\boldsymbol{y} \in \mathbb{R}^d$ which minimizes the following energy function:

$$\mathbf{E}(\boldsymbol{z}, \boldsymbol{y}, \mathcal{D}) = \frac{1}{2}\|\boldsymbol{y} - \mathcal{D}\boldsymbol{z}\|_2^2 + \lambda\|\boldsymbol{z}\|_1. \tag{1}$$

The first term in (1) is the reconstruction error for the sample $\boldsymbol{y}$ using code $\boldsymbol{z}$ and decoder $\mathcal{D}$. The second term is a regularization term which penalizes the l_1 norm of $\boldsymbol{z}$. In essence, each code $\boldsymbol{z}$ selects a linear combination of the decoder's columns. The decoder can thus be thought of as a *dictionary* of building blocks. Inference algorithms find a solution for the minimization problem in (1) keeping $\mathcal{D}$ fixed, namely

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \mathbf{E}(\boldsymbol{z}, \boldsymbol{y}, \mathcal{D}).$$

Learning of the Decoder The decoder's parameters can be learned through gradient-based optimization by minimizing the mean squared reconstruction error between elements in the training set $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subset \mathbb{R}^d$ and their corresponding reconstructions from the sparse representations $\mathcal{Z}^* = \{\boldsymbol{z}_1^*, \ldots, \boldsymbol{z}_N^*\} \subset \mathbb{R}^l$ obtained during inference, namely:

$$\arg\min_{\mathcal{D}} \mathcal{L}_{\mathcal{D}}(\mathcal{D}, \mathcal{Z}^*, \mathcal{Y}) = \arg\min_{\mathcal{D}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{y}_i - \mathcal{D}\boldsymbol{z}_i^*\|_2^2. \tag{2}$$

2.2 Background: Inference with ISTA and FISTA

We provide an overview of the Iterative Shrinkage-Thresholding Algorithm (ISTA) and its faster version FISTA which we adopt in our setup. ISTA is a proximal gradient method which performs inference to find a sparse code $\boldsymbol{z}^*$ which minimizes the energy in (1) for a given input $\boldsymbol{y}$ and a fixed dictionary $\mathcal{D}$. We can re-write the inference minimization problem as:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \mathbf{E}(\boldsymbol{z}, \boldsymbol{y}, \mathcal{D}) = \arg\min_{\boldsymbol{z}} f(\boldsymbol{z}) + g(\boldsymbol{z}), \tag{3}$$

where $f(\boldsymbol{z}) = \frac{1}{2}\|\boldsymbol{y} - \mathcal{D}\boldsymbol{z}\|_2^2$ is the reconstruction term and $g(\boldsymbol{z}) = \lambda\|\boldsymbol{z}\|_1$ is the l_1 penalty term. In ISTA, the code $\boldsymbol{z}$ can be given any initial value and is typically set to the zero vector: $\boldsymbol{z}^{(0)} = \mathbf{0} \in \mathbb{R}^l$. An ISTA iteration consists of two steps: a gradient step and a shrinkage step.

Gradient Step To compute code $\boldsymbol{z}^{(k)}$ from its previous iteration $\boldsymbol{z}^{(k-1)}$, where $k \in \{1, \ldots, K\}$, ISTA first performs the gradient update by taking gradient step of size η_k with respect to the squared error between the input and the reconstruction from $\boldsymbol{z}^{(k-1)}$:

$$\text{ISTA gradient step: } \tilde{\boldsymbol{z}}^{(k)} = \boldsymbol{z}^{(k-1)} - \eta_k \nabla f(\boldsymbol{z}^{(k-1)}), \tag{4}$$

The size of the gradient step η_k at iteration k can be determined with backtracking (Beck & Teboulle, 2009).

Shrinkage Step The shrinkage step is applied to the intermediate output from the gradient step in ISTA. It uses the shrinkage function τ_α which sets components of its input to 0 if their absolute value is less than α or otherwise contracts them by α, namely $[\tau_\alpha(\boldsymbol{x})]_j = \text{sign}(x_j)(|x_j| - \alpha)_+$ for some threshold $\alpha > 0$ where j traverses the components of its input vector $\boldsymbol{x}$. The shrinkage step for ISTA is given by:

$$\text{ISTA shrinkage step: } \boldsymbol{z}^{(k)} = \tau_{\lambda\eta_k}(\tilde{\boldsymbol{z}}^{(k)}) \tag{5}$$

where $\lambda > 0$ is a hyperparameter which determines the sparsity level. In the case when the codes are restricted to be non-negative, the shrinkage function can be modified to set all negative components in its output to 0: $[\tilde{\tau}_\alpha(\boldsymbol{x})]_j = ([\tau_\alpha(\boldsymbol{x})]_j)_+$.

FISTA In our setup, we use FISTA (Beck & Teboulle, 2009) which accelerates the convergence of ISTA by adding a momentum term in each of its iterations. Namely, $\boldsymbol{z}^{(k-1)}$ in (4) is replaced by:

$$\boldsymbol{x}^{(k)} = \boldsymbol{z}^{(k-1)} + \frac{t_{k-1} - 1}{t_k}(\boldsymbol{z}^{(k-1)} - \boldsymbol{z}^{(k-2)})$$

for $k \geq 2$ where $t_k = \frac{1+\sqrt{1+4t_{k-1}^2}}{2}$ and $t_1 = 1$. Thus, the FISTA gradient step becomes:

$$\tilde{\boldsymbol{z}}^{(k)} = \boldsymbol{x}^{(k)} - \eta_k \nabla f(\boldsymbol{x}^{(k)}). \tag{6}$$

FISTA employs the same shrinkage step as ISTA.

2.3 Modified Inference with Variance Regularization on the Latent Code Components

In order to prevent collapse in the l_1 norm of the latent codes, we propose to ensure that the variance of each latent code component stays above a pre-set threshold. To achieve this, we add a regularization term to the energy in (3) which encourages the variance of all latent component across a mini-batch of codes to be greater than a pre-set threshold. In particular, we replace the function $f(\boldsymbol{z}) = \frac{1}{2}\|\boldsymbol{y} - \mathcal{D}\boldsymbol{z}\|_2^2$ in (3) with a new function $\tilde{f}(\boldsymbol{Z})$ defined over a set of codes $\boldsymbol{Z} \in \mathbb{R}^{l \times n}$ corresponding to a mini-batch of data samples $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$ as:

$$\tilde{f}(\boldsymbol{Z}) = \sum_{i=1}^{n} \frac{1}{2}\|\boldsymbol{y}_i - \mathcal{D}\boldsymbol{Z}_{:,i}\|_2^2 + \sum_{j=1}^{l} \beta\Big[\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{j,:})}\Big)_+\Big]^2. \tag{7}$$

The first sum in (7) is over the reconstruction terms from each code $\boldsymbol{Z}_{:,i} \in \mathbb{R}^l$. The second sum in (7) is over squared hinge terms involving the variance of each latent component $\boldsymbol{Z}_{j,:} \in \mathbb{R}^n$ across the batch where $\text{Var}(\boldsymbol{Z}_{j,:}) = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{Z}_{j,i} - \mu_j)^2$ and μ_j is the mean across the j-th latent component, namely $\mu_j = \frac{1}{n}\sum_{i=1}^{n} Z_{j,i}$. The hinge terms are non-zero for any latent dimension whose variance is below the fixed threshold of $\sqrt{T}$.

Modified Energy: Variance Regularization Using $\tilde{f}(\boldsymbol{Z})$ from (7) and setting $\tilde{g}(\boldsymbol{Z}) = \sum_{i=1}^{n} g(\boldsymbol{Z}_{:,i})$, our modified objective during inference is to minimize the energy:

$$\tilde{\mathbf{E}}(\boldsymbol{Z}, Y, \mathcal{D}) = \tilde{f}(\boldsymbol{Z}) + \tilde{g}(\boldsymbol{Z}) = \sum_{i=1}^{n} \frac{1}{2}\|\boldsymbol{y}_i - \mathcal{D}\boldsymbol{Z}_{:,i}\|_2^2 + \sum_{j=1}^{l} \beta\Big[\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{j,:})}\Big)_+\Big]^2 + \sum_{i=1}^{n} \lambda\|\boldsymbol{Z}_{:,i}\|_1 \tag{8}$$

with respect to the codes $\boldsymbol{Z}$ for a mini-batch of samples Y and a fixed dictionary $\mathcal{D}$. Note that the hinge terms counteract with the l_1 penalty terms in (8). The hinge terms act as regularization terms which encourage the variance of each of the latent code components to remain above the threshold of $\sqrt{T}$. This should prevent a collapse in the l_1 norm of the latent codes and thus should remove the need to normalize the weights of the decoder.

Gradient Step The first step in our modified version of FISTA is to take a gradient step for each code $\boldsymbol{Z}_{:,t} \in \mathbb{R}^l$:

$$\tilde{\boldsymbol{Z}}_{:,t}^{(k)} = \boldsymbol{Z}_{:,t}^{(k-1)} - \eta_k \nabla_{\boldsymbol{Z}_{:,t}^{(k-1)}} \tilde{f}(\boldsymbol{Z}).^1 \tag{9}$$

The gradient of the sum of reconstruction terms in (7) with respect to $Z_{s,t}$, one of the latent components of code $\boldsymbol{Z}_{:,t}$, for a linear decoder $\mathcal{D}$ is:

$$\frac{\partial}{\partial Z_{s,t}} \sum_{i=1}^{n} \frac{1}{2} \|\boldsymbol{y}_i - \mathcal{D}\boldsymbol{Z}_{:,i}\|_2^2 = \mathcal{D}_s^T(\mathcal{D}\boldsymbol{Z}_{:,t} - \boldsymbol{y}_t), \tag{10}$$

where $\mathcal{D}_s$ denotes the s-th column of the dictionary $\mathcal{D}$.[2] The gradient of the sum of hinge terms in (7) with respect to $Z_{s,t}$ is:

$$\frac{\partial}{\partial Z_{s,t}} \sum_{j=1}^{l} \beta \Big[\Big(T - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{j,:})} \Big)_+ \Big]^2 = \begin{cases} -\frac{2\beta}{n-1} \frac{T - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})}}{\sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})}} (Z_{s,t} - \mu_s), & \text{if } \sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})} < T \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Even though the hinge terms in (7) are not smooth convex functions[3], the fact that the gradient in (11) is a line implies that the hinge term behaves locally as a convex quadratic function. A visualization of $h(\boldsymbol{x}) = \big[\big(1 - \sqrt{\mathrm{Var}(\boldsymbol{x})} \big)_+ \big]^2$ for $\boldsymbol{x} \in \mathbb{R}^2$ and the full derivation of (11) are available in Appendix A.3 and A.3.1, respectively.

Shrinkage Step Similarly to the regular FISTA protocol, the shrinkage step in our modified FISTA is applied to the intermediate results from the gradient step in (9).

2.4 Encoder for Amortized Inference

We propose to train an encoder $\mathcal{E}$ simultaneously with the decoder $\mathcal{D}$ to predict the sparse codes computed through inference with FISTA. The first reason is to avoid using batch statistics for computing the codes once the decoder is trained. Indeed, it should be possible to compute codes for different inputs independently from each other. The second reason is to reduce the inference time. After training of the encoder and decoder is done, the encoder can compute the sparse representations of inputs directly which removes the need to perform inference with FISTA, i.e. the encoder enables *amortized inference*.

Learning of the Encoder The encoder is trained using mini-batch gradient descent to minimize the following mean-squared error between its predictions and the sparse codes $\boldsymbol{Z}^* \in \mathbb{R}^{l \times n}$ computed from inference with FISTA for inputs $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$:

$$\arg\min_{\mathcal{E}} \mathcal{L}_{\mathcal{E}}(\mathcal{E}, \boldsymbol{Z}^*, Y) = \arg\min_{\mathcal{E}} \frac{1}{n} \sum_{i=1}^{n} \|\mathcal{E}(\boldsymbol{y}_i) - \boldsymbol{Z}_{:,i}^*\|_2^2. \tag{12}$$

Note that the codes $\boldsymbol{Z}^*$ are treated as constants. Training the encoder this way is similar to target propagation (Lee et al., 2015).

Modified Energy: Encoder Regularization To ensure that the codes computed during inference with FISTA do not deviate too much from the encoder's predictions, we further modify the inference protocol by adding a term which penalizes the distance between the encoder's outputs and FISTA's outputs:

$$\tilde{f}(\boldsymbol{Z}) = \sum_{i=1}^{n} \frac{1}{2} \|\boldsymbol{y}_i - \mathcal{D}\boldsymbol{Z}_{:,i}\|_2^2 + \beta \sum_{j=1}^{l} \Big[\Big(T - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{j,:})} \Big)_+ \Big]^2 + \gamma \sum_{i=1}^{n} \|\boldsymbol{Z}_{:,i} - \mathcal{E}(\boldsymbol{y}_i)\|_2^2. \tag{13}$$

This regularization term encourages FISTA to find codes which can be learned by the encoder. In our setup, the encoder's predictions are treated as constants and are used as the initial value for codes in FISTA, namely $\boldsymbol{Z}_{:,i}^{(0)} = \mathcal{E}(\boldsymbol{y}_i)$. This can reduce the inference time by lowering the number of FISTA iterations needed for convergence if the encoder provides a good initial values.

[1] For simplicity, we do not use the momentum term notation introduced in (6).

[2] The gradient for any neural network decoder $\mathcal{D}$ can be computed through automatic differentiation.

[3] A requirement for the FISTA algorithm.

3 Experimental Setup

3.1 Encoder and Decoder Architectures

LISTA Encoder The encoder's architecture is inspired by the Learned ISTA (LISTA) architecture (Gregor & LeCun, 2010). LISTA is designed to mimic outputs from inference with ISTA and resembles a recurrent neural network. Our encoder consists of two fully connected layers $\boldsymbol{U} \in \mathbb{R}^{d \times l}$ and $\boldsymbol{S} \in \mathbb{R}^{l \times l}$, a bias term $\boldsymbol{b} \in \mathbb{R}^l$, and ReLU activation functions. Algorithm 1 describes how the encoder's outputs are computed.

Algorithm 1 LISTA encoder $\mathcal{E}$

Input: Image $\boldsymbol{y} \in \mathbb{R}^d$, number of iterations L
Parameters: $\boldsymbol{U} \in \mathbb{R}^{d \times l}$, $\boldsymbol{S} \in \mathbb{R}^{l \times l}$, $\boldsymbol{b} \in \mathbb{R}^l$
Output: sparse code $\boldsymbol{z}_\mathcal{E} \in \mathbb{R}^l$
$\boldsymbol{u} = \boldsymbol{U}\boldsymbol{y} + \boldsymbol{b}$
$\boldsymbol{z}_0 = \mathrm{ReLU}(\boldsymbol{u})$
for $i = 1$ **to** L **do**
  $\boldsymbol{z}_i = \mathrm{ReLU}(\boldsymbol{u} + \boldsymbol{S}\boldsymbol{z}_{i-1})$
end for
$\boldsymbol{z}_\mathcal{E} = \boldsymbol{z}_L$

Linear Decoder A linear decoder $\mathcal{D}$ is parameterized simply as a linear transformation $\boldsymbol{W}$ which maps codes in $\mathbb{R}^l$ to reconstructions in the input data dimension $\mathbb{R}^d$, i.e. $\boldsymbol{W} \in \mathbb{R}^{d \times l}$. No bias term is used in this linear transformation.

Non-Linear Decoder In the case of a non-linear decoder $\mathcal{D}$, we use a fully connected network with one hidden layer of size m and input layer of size l (the dimension of the latent codes). We use ReLU as an activation function in the hidden layer. For convenience, we refer to the layer which maps the input codes to the hidden representation as $\boldsymbol{W}_1 \in \mathbb{R}^{m \times l}$ and the one which maps the hidden representation to the output (reconstructions) as $\boldsymbol{W}_2 \in \mathbb{R}^{d \times m}$. There is a bias term $\boldsymbol{b}_1 \in \mathbb{R}^m$ following $\boldsymbol{W}_1$ and no bias term after $\boldsymbol{W}_2$.

3.2 Evaluation

We compare autoencoders trained using our variance regularization method to autoencoders trained with the traditional sparse coding approach in which the l_2 norm of the decoder's columns is fixed. We refer to the traditional approach as *standard dictionary learning* (**SDL**) in the case the decoder is a linear dictionary and as **SDL-NL** in the case it is non-linear - a natural extension of SDL to the case of non-linear decoders is to fix the l_2 norms of the columns of both $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. We refer to our approach as *variance-regularized dictionary learning* (**VDL**) in the case when the decoder is linear and as **VDL-NL** when the decoder is non-linear. The weights of the decoders in VDL and VDL-NL models are not restricted.

In order to compare the autoencoders, we train them with different levels of sparsity regularization λ, evaluate the quality of their reconstructions in terms of peak signal-to-noise ratio (PSNR), and visualize the features that they learn. We evaluate models with a linear decoder on the downstream task of denoising. Additionally, we measure the pre-trained models' performance on the downstream task of MNIST classification in the low data regime.

3.3 Data Processing

MNIST In the first set of our experiments, we use the MNIST dataset (LeCun & Cortes, 2010) consisting of 28×28 hand-written digits and do standard pre-processing by subtracting the global mean and dividing by the global standard deviation. We split the training data randomly into 55000 training samples and 5000 validation samples. The test set consists of 10000 images.

Natural Image Patches For experiments with natural images, we use patches from ImageNet ILSVRC-2012 (Deng et al., 2009). We first convert the ImageNet images to grayscale and then do standard pre-processing by subtracting the global mean and dividing by the global standard deviation. We then apply local contrast normalization to the images with a 13×13 Gaussian filter with standard deviation of 5 following Jarrett et al. (2009); Zeiler et al. (2011). We use 200000 randomly selected patches of size 28×28 for training, 20000 patches for validation, and 20000 patches for testing.

3.4 Inference and Training

We determine all hyperparameter values through grid search. Full training details can be found in Appendix A.2.

Inference The codes $\boldsymbol{Z}$ are restricted to be non-negative as described in section 2.2. The dimension of the latent codes in our MNIST experiments is $l = 128$ and it is $l = 256$ for experiments with ImageNet patches. The batch size is set to 250 which we find sufficiently large for the regularization term on the variance of each latent component in VDL in (7). We set the maximum number of FISTA iterations K to 200 which we find sufficient for good reconstructions.

Autoencoders Training We train models for a maximum of 200 epochs in MNIST experiments and for 100 epochs in experiments with natural image patches. We apply early stopping and save the autoencoder which outputs codes with the lowest average energy measured on the validation set.[4] In SDL and SDL-NL experiments, we fix the l_2 norms of the columns in the decoders' fully connected layers $\boldsymbol{W}$, $\boldsymbol{W}_1$, and $\boldsymbol{W}_2$ to be equal to 1. We add weight decay to the bias term $\boldsymbol{b}_1$ in both SDL-NL and VDL-NL models as well as to the bias term $\boldsymbol{b}$ in the LISTA encoder to prevent their norms from inflating arbitrarily.

Implementation and Hardware Our PyTorch implementation is available on GitHub at [link removed for anonymity]. We train our models on one NVIDIA RTX 8000 GPU card and all our experiments take less than 24 hours to run.

4 Results and Analysis

4.1 Autoencoders with a Linear Decoder

Linear Decoders: Features Figures 2 shows the dictionary elements for two SDL and two VDL decoders trained with different levels of sparsity regularization λ. As evident from Figures 2a and 2b we confirm that, in the case of standard dictionary learning on MNIST, the dictionary atoms look like orientations, strokes, and parts of digits for lower values of the sparsity regularization λ and they become looking like full digit prototypes as λ increases (Yu, 2012). A similar transition from strokes to full digit prototypes is observed in figures 2c and 2d displaying the dictionary elements of models trained using variance regularization in the latent code components.

Figure 3 shows dictionary elements for two SDL and two VDL models trained on ImageNet patches. Decoders in both SDL and VDL models contain Gabor-like filters. We observe that models which are trained with a lower sparsity regularization parameter λ (Figure 3a and 3c) contain more high frequency atoms than ones trained with higher values of λ (Figure 3b and 3d).

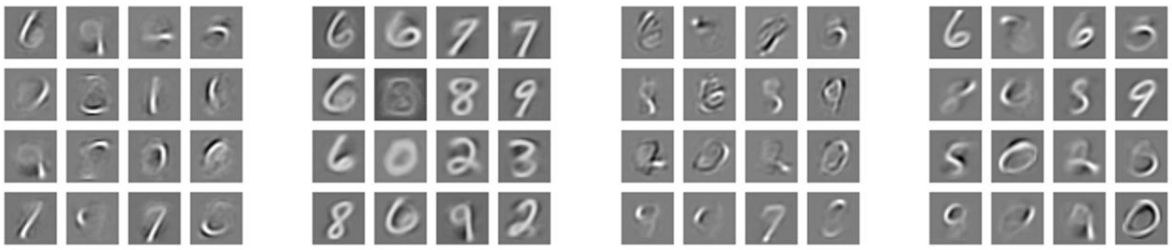
LISTA and Linear Decoders: Reconstruction Quality The dashed lines in Figure 4a represent regret curves for encoders from SDL and VDL models (with a linear decoder). These regret curves show the trade-off between the level of sparsity measured by the average percentage of non-active (zero) code components and reconstruction quality measured by the average PSNR evaluated on the test set over 5 random seeds. As expected, higher sparsity levels result in worse reconstructions. Models trained with our variance regularization approach produce better reconstructions than SDL models for higher levels of sparsity.[5]

LISTA and Linear Decoders: Denoising We evaluate the performance of the encoders trained to predict sparse codes in SDL and VDL models on the downstream task of denoising. In this setup, we corrupt the input images by adding Gaussian noise and proceed with computing their corresponding sparse codes through amortized inference using the encoder. We then feed these codes to the decoder to obtain reconstructions of the noisy inputs.

Figure 5 shows the reconstructions of original MNIST images and of MNIST images corrupted with Gaussian noise with zero mean standard deviation $\sigma = 1$ using encoders trained with the SDL and VDL dictionaries

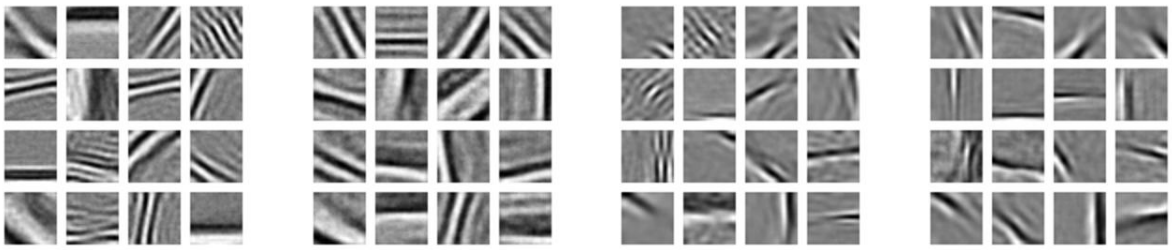
[4] As a reminder, codes are computed using amortized inference with the encoder during validation.

[5] As a reference for reconstruction quality in terms of PSNR, the second row of images in Figure 5a and 5b contains reconstructions of the images in the top row with PSNR of around 17.3 and 17.7, respectively.



(a) SDL, $\lambda = 0.001$ (b) SDL, $\lambda = 0.005$ (c) VDL, $\lambda = 0.005$ (d) VDL, $\lambda = 0.02$

Figure 2: Dictionary elements for linear dictionaries with latent dimension $l = 128$ and different levels of sparsity regularization λ. The SDL model in (2a) produces reconstructions with PSNR of 21.1 for codes with average sparsity level of 63% on the test set. The VDL model in (2c) produces reconstructions with PSNR of 20.7 for codes with average sparsity level of 69%. The SDL model in (2b) produces reconstructions with PSNR of 18.6 for codes with average sparsity level of 83%. The VDL model in (2d) produces reconstructions with PSNR of 17.7 for codes with average sparsity level of 91.8%.



(a) SDL, $\lambda = 0.002$ (b) SDL, $\lambda = 0.005$ (c) VDL, $\lambda = 0.005$ (d) VDL, $\lambda = 0.01$

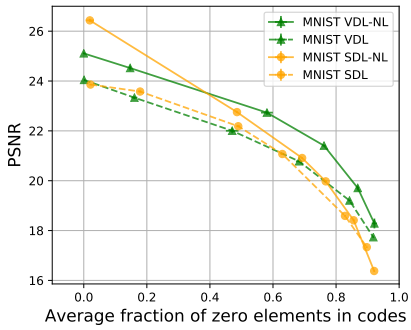
Figure 3: Dictionary elements which resemble Gabor filters for SDL and VDL models with latent dimension $l = 256$ trained on ImageNet patches with different levels of sparsity regularization λ. The SDL models in 3a and 3b produce reconstructions with average PSNR of 26.9 and 24.0 and average sparsity level in the codes of 74.6% and 90.5% on the test set, respectively. The VDL models in Figures 3c and 3d produce reconstructions with average PSNR of 27.3 and 25.3 and average sparsity level in the codes of 77.3% and 89.2% on the test set, respectively.

from Figure 2b and 2d. We observe that both SDL and VDL models are able to produce reconstructions which are closer to samples from the manifold of training data than to the noisy inputs. This implies that the sparse coding systems have learned to detect features useful for the data they are trained on and cannot reconstruct random noise, as desired.

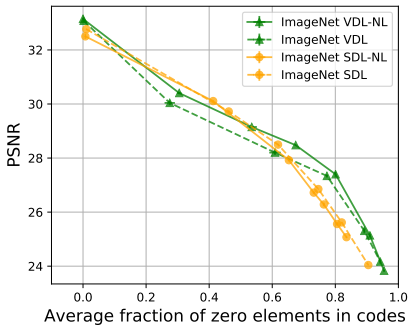
SDL and VDL models trained on ImageNet patches are also robust to the addition of Gaussian noise to their inputs. This is evident in Figure 6 which contains reconstructions $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}_\sigma$ of images $\boldsymbol{y}$ and their noisy version $\boldsymbol{y}_\sigma$ for an SDL model and a VDL model with similar average sparsity in the codes they produce.

Table 1 shows a summary of the reconstruction quality of images with varying levels of Gaussian noise corruption for encoders from SDL and VDL models over 5 random seeds. Both the SDL and VDL encoders are able to produce codes from corrupted inputs $\boldsymbol{y}_\sigma$ which, when passed through their corresponding decoders, produce outputs $\tilde{\boldsymbol{y}}_\sigma$ that are less noisy than the corrupted inputs. This observation is significant considering the fact that the sparse autoencoders in our experiments are not explicitly trained to perform denoising.

Ablation: no variance regularization In experiments without applying our variance regularization strategy on the sparse representations or fixing the norms of the dictionary's columns, we observe a collapse in the l_1 norms of the sparse codes. This indicates that VDL is an effective alternative strategy to SDL for preventing collapse.

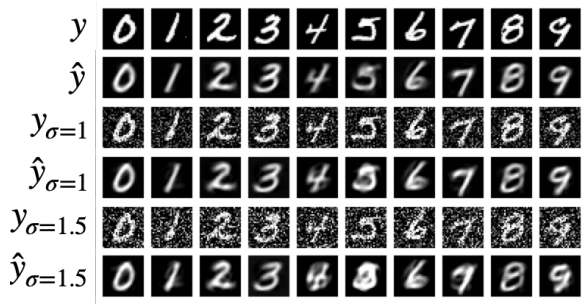


(a) MNIST

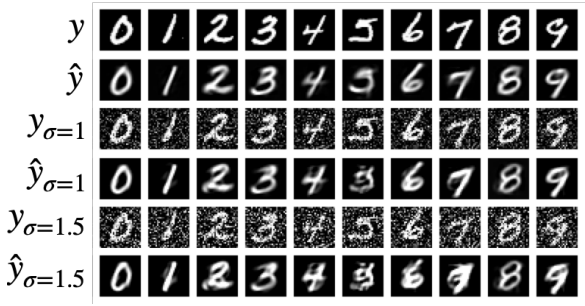


(b) ImageNet patches

Figure 4: Trade-off between sparsity measured by average percentage of inactive (zero) code components and reconstruction quality measured by PSNR for encoders from SDL, SDL-NL, VDL, and VDL-NL models trained on MNIST with code dimension $l = 128$ in (4a) or on ImageNet patches with code dimension $l = 256$ in (4b) and different levels of sparsity regularization λ. Higher PSNR reflects better reconstruction quality.



(a) SDL, $\lambda = 0.005$



(b) VDL, $\lambda = 0.02$

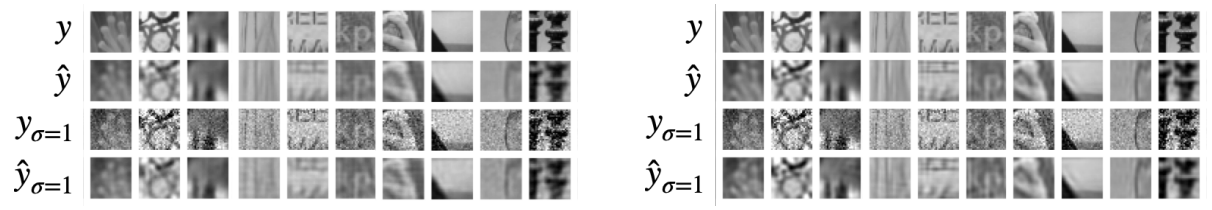
Figure 5: Examples of denoising abilities of encoders from SDL and VDL models trained on MNIST which produce codes with average sparsity of 89.7% and 91.8%, respectively. Top two rows: original inputs (top) and their reconstructions (bottom); middle two rows: inputs with Gaussian noise with standard deviation $\sigma = 1$ (top) and their reconstructions (bottom), bottom two rows: inputs with Gaussian noise with standard deviation $\sigma = 1.5$ (top) and their reconstructions (bottom).

4.1.1 Autoencoders with a Non-linear Decoder

Non-linear Decoders: Features Each column $\boldsymbol{W}_{:,i} \in \mathbb{R}^d$ in a linear dictionary $\mathcal{D}$ (parameterized by $\boldsymbol{W}$) reflects what a single latent component encodes since $\boldsymbol{W}_{:,i} = W\boldsymbol{e}^{(i)}$ where $\boldsymbol{e}^{(i)} \in \mathbb{R}^l$ is a one-hot vector with 1 in position i and 0s elsewhere. We use a similar strategy to visualize the features that non-linear decoders learn. Namely, we provide codes with only one non-zero component as inputs to the non-linear decoders and display the resulting "atoms".[6] While in the linear case each code component selects a single dictionary column, in the case of non-linear decoders with one hidden layer, each code component $\boldsymbol{e}^{(i)}$ selects a linear combination of columns in W_2. Two SDL-NL and two VDL-NL models trained on MNIST with different levels of sparsity regularization λ yield the atoms displayed in Figure 7. They contain strokes and orientations for smaller values of λ and fuller digit prototypes for larger values of λ which are very similar to the dictionary elements in Figure 2 for both the SDL-NL and VDL-NL models.

LISTA and Non-linear Decoders: Reconstruction Quality Figure 4 plots the trade-off between average sparsity in the codes (computed by encoders) and reconstruction quality from these codes for models with linear decoders (dashed lines) and non-linear decoders (solid lines). Figures 4a and 4b show that VDL-NL models trained on both MNIST and ImageNet patches have better PSNR performance than all the other

[6]Additionally, we subtract the effect of the bias. In essence, we display $\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{e}^{(i)}$ rather than $\boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{e}^{(i)} + \boldsymbol{b}_1)$.



(a) SDL, $\lambda = 0.002$ (b) VDL, $\lambda = 0.005$

Figure 6: Examples of denoising for encoders from SDL and VDL models trained on ImageNet patches which produce codes with average sparsity of 74.6% and 77.3%, respectively. Top two rows: original inputs (top) and their reconstructions (bottom); bottom two rows: inputs corrupted with Gaussian noise with standard deviation $\sigma = 1$ (top) and their reconstructions (bottom).

Table 1: The performance of SDL and VDL models trained on MNIST and ImageNet patches with different levels of sparsity regularization λ on the the task of denoising inputs corrupted with Gaussian noise with standard deviation $\sigma = 1.0$ or $\sigma = 1.5$. We report the average sparsity of codes computed with these models and the PSNR of: the reconstruction $\tilde{\boldsymbol{y}}$ of the original (not corrupted) input $\boldsymbol{y}$, the corrupted input $\boldsymbol{y}_\sigma$, and the reconstruction $\tilde{\boldsymbol{y}}_\sigma$ of the corrupted input $\boldsymbol{y}_\sigma$. Evaluated on the test set over 5 random seeds.

MNIST							
Model	λ	$L_0(\boldsymbol{z}_{\tilde{\boldsymbol{y}}})$	PSNR $\tilde{\boldsymbol{y}}$	σ	PSNR $\boldsymbol{y}_\sigma$	$L_0(\boldsymbol{z}_{\tilde{\boldsymbol{y}}_\sigma})$	PSNR $\tilde{\boldsymbol{y}}_\sigma$
SDL	0.005	$89.7\% \pm 0.2\%$	17.3 ± 0.028	1.0	10.2	$87.3\% \pm 0.2\%$	16.9 ± 0.053
VDL	0.02	$91.8\% \pm 0.1\%$	17.7 ± 0.023	1.0	10.2	$88.1\% \pm 0.2\%$	15.4 ± 0.175
SDL	0.005	$89.7\% \pm 0.2\%$	17.3 ± 0.028	1.5	6.7	$84.6\% \pm 0.5\%$	15.7 ± 0.204
VDL	0.02	$91.8\% \pm 0.1\%$	17.7 ± 0.023	1.5	6.7	$84.5\% \pm 0.3\%$	12.2 ± 0.275
ImageNet patches							
SDL	0.002	$74.6\% \pm 0.1\%$	26.9 ± 0.007	1.0	20.0	$72.7\% \pm 0.1\%$	25.6 ± 0.011
VDL	0.005	$77.3\% \pm 0.2\%$	27.3 ± 0.058	1.0	20.0	$73.5\% \pm 0.2\%$	25.2 ± 0.035

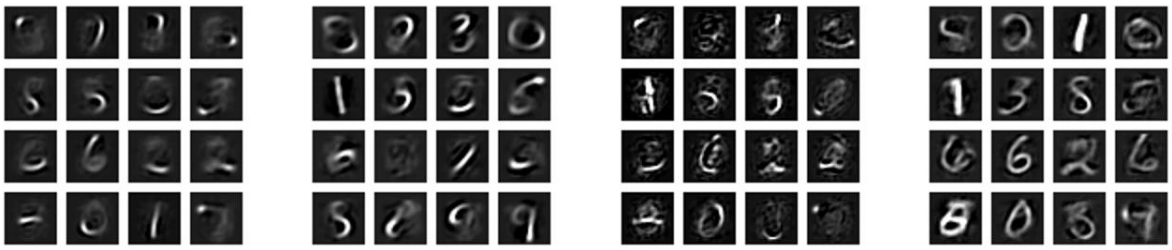
models for high levels of sparsity. This implies that models with non-linear decoders trained using variance regularization on the codes can better utilize the additional representational capacity from the larger number of layers and parameters in $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ compared to the other models when the sparsity level is high.

Non-Linear Decoders: Features As described in the case for MNIST in section 4.1, we visualize the features that non-linear decoders learn by displaying what each latent code component encodes. These learned features are displayed in Figure 8 for SDL-NL and VDL-NL models trained with different levels of sparsity regularization λ. As in the case of linear decoders, we observe that both SDL-NL and VDL-NL models learn Gabor filters.

Classification in the Low Data Regime To investigate whether self-supervised pre-training helps with classification performance in the low data regime, we evaluate the linear separability of codes from pre-trained LISTA encoders in SDL, SDL-NL, VDL, and VDL-NL autoencoders[7] when few training samples are available for supervised learning. To do this, we freeze the pre-trained encoders' weights and train a linear classification layer on top of their output layer with 1, 2, 5, 10, 20, 50, and 100 training samples per class. For each number of training samples per class and each type of encoder (coming from SDL, VDL, SDL-NL, or VDL-NL models), we select the encoder among the ones presented in Figure 4a which gives the best validation accuracy.

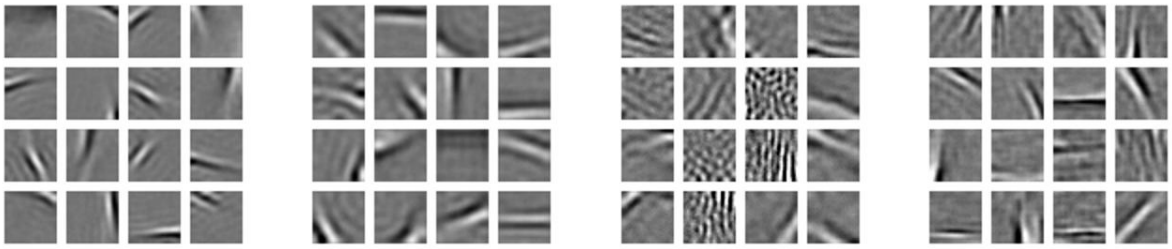
We compare the accuracy of the linear classifiers trained on the features output by the frozen pre-trained encoders to that of classifiers trained from scratch on the raw MNIST data. We consider a linear classifier (referred to as "Linear on raw") and a classifier consisting of a LISTA encoder followed by a linear layer

[7]The autoencoders are trained on the full MNIST dataset in an unsupervised way.



(a) SDL-NL, $\lambda = 0.003$ (b) SDL-NL, $\lambda = 0.01$ (c) VDL-NL, $\lambda = 0.003$ (d) VDL-NL, $\lambda = 0.02$

Figure 7: Visualizing reconstructions from SDL-NL and VDL-NL decoders trained on MNIST with input codes in which only one component is active and the rest are zero. We observe that for both SDL-NL and VDL-NL, each code component encodes stokes, orientations, parts of digits, and full digit prototypes very similar to those in Figure 2. The SDL-NL models in 7a and 7b produce reconstructions with average PSNR of 20.9 and 18.4 and average sparsity level in the codes of 69.2% and 85.6% on the test set, respectively. The VDL-NL models in 7c and 7d produce reconstructions with average PSNR of 22.7 and 18.3 and average sparsity level in the codes of 58.8% and 92.2% on the test set, respectively.



(a) SDL-NL, $\lambda = 0.003$ (b) SDL-NL, $\lambda = 0.01$ (c) VDL-NL, $\lambda = 0.003$ (d) VDL-NL, $\lambda = 0.01$

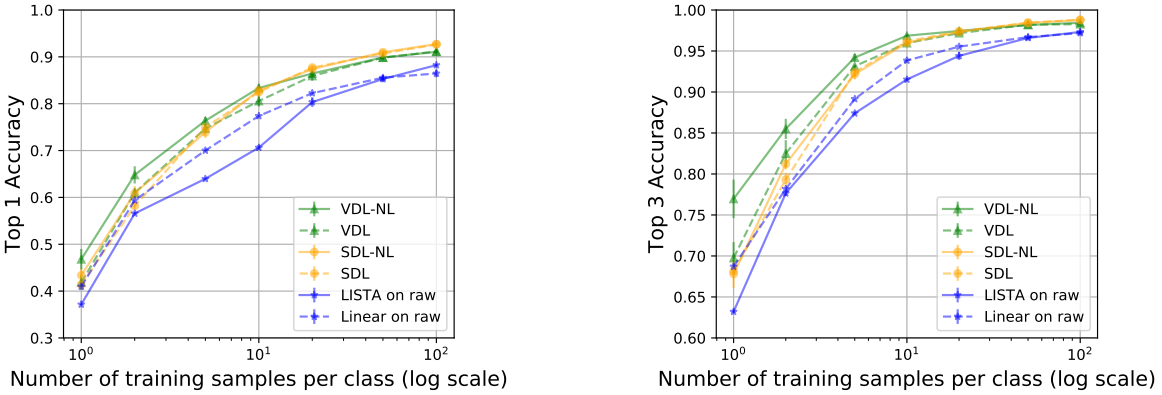
Figure 8: Visualizing reconstructions produced by SDL-NL and VDL-NL decoders trained on ImageNet patches from input codes in which only one code component is active and the rest are zero. We observe that for both SDL-NL and VDL-NL, code components encode Gabor-like filters. The SDL-NL and VDL-NL models in (8a) and (8c) produce reconstructions with PSNR of 27.1 and 28.4 for codes with average sparsity level of 69.8% and 67.6%, respectively. The SDL-NL and VDL-NL models in (8b) and (8d) produce reconstructions with PSNR of 25.0 and 24.9 for codes with average sparsity level of 83.4% and 90.8%, respectively.

(referred to as "LISTA on raw"). Figure 9 summarizes the classification performance on the test set. All linear classifiers trained on top of the pre-trained LISTA encoders give better top 1 and top 3 classification performance than classifiers trained on the raw MNIST data. Furthermore, VDL-NL models outperform VDL, SDL, and SDL-NL models on classification with up to 10 samples per class. This implies that sparse autoencoders which use a non-linear decoder and are trained using variance regularization on the latent codes give better classification performance when few labeled training samples available.

Ablation: no variance regularization Similarly to the linear case, experiments in which $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are unconstrained and the standard FISTA (without variance regularization applied to the codes) is used result in a collapse of the l_1 norm of the latent codes. Our experiments imply that our proposed variance regularization protocol prevents collapse in models which use fully-connected decoders with one hidden layer.

5 Related Work

Our work is related to the extensive literature on sparse coding and dictionary learning. Mairal et al. (2014) provide a comprehensive overview of sparse modeling in computer vision, discuss its relation to the fields of



(a) Top 1 accuracy.

(b) Top 3 accuracy.

Figure 9: Linear separability of features from pre-trained SDL, SDL-NL, VDL, and VDL-NL encoders. Models trained with variance regularization in the codes and a non-linear decoder (VDL-NL) outperform other models in linear classification with up to 10 training samples per class. Test set performance is reported over 5 random seeds.

statistics, information theory, and signal processing, as well as its applications in computer vision including image denoising, inpainting, and super-resolution.

Sparsity Constraints Sparse representations can be obtained using many different objectives. The l_1 penalty term in equation 1 can be replaced by other sparsity-inducing constraints including l_0, l_p $(0 < p < 1)$, l_2, or $l_{2,1}$ norms. In particular, the l_1 term is a convex relaxation of the non-differentiable l_0 constraint of the form $\|\boldsymbol{z}\|_0 \leq k$ requiring $\boldsymbol{z}$ to have at most k active components. Zhang et al. (2015) provide a survey of existing algorithms for sparse coding such as K-SVD which generalizes K-means clustering (Aharon et al., 2006). Makhzani & Frey (2013) propose an approach to train a sparse autoencoder with a linear encoder and decoder in which only the top k activations in the hidden layer are kept and no other sparsifying penalty is used.

Inference There are many variations of the ISTA inference algorithm which address the fact that it is computationally expensive and are designed to speed it up, such as FISTA (Beck & Teboulle, 2009) which is used in our setup (for details please refer to section 2.2). Gated LISTA (Wu et al., 2019) introduces a novel gated mechanism for LISTA (Gregor & LeCun, 2010) which enhanses its performance. In Zhou et al. (2018), it is shown that deep learning with LSTMs can be used to improve learning sparse codes in ISTA by modeling the history of the iterative algorithm and acting like a momentum. Inspired by LISTA, we train an encoder to predict the representations computed during inference using the usual and modified FISTA protocols as explained in section 2.4.

Multi-layer Sparse Coding There exist methods which greedily construct hierarchies of sparse representations. Zeiler et al. (2010) use ISTA in a top-down convolutional multi-layer architecture to learn sparse latent representations at each level in a greedy layer-wise fashion. Other convolutional models which produce a hierarchy of sparse representations are convolutional DBNs (Lee et al., 2009) and hierarchical convolutional factor analysis (Chen et al., 2013) which also use layer-wise training. Another greedy approach is proposed by Tariyal et al. (2016) who train multiple levels of dictionaries, one layer at a time. In contrast, our work is about finding sparse representations by utilizing multi-layer neural networks and training the whole system end-to-end.

6 Conclusion

In this work, we revisit the traditional setup of sparse coding with l_1 sparsity penalty. We propose to apply variance regularization to the sparse codes during inference with the goal of training non-linear decoders

without collapse in the l_1 norm of the codes. We show that using our proposed method we can successfully train sparse autoencoders with fully connected multi-layer decoders which have interpretable features, outperform models with linear decoders in terms of reconstruction quality for a given average level of sparsity in the codes, and improve MNIST classification performance in the low data regime. Future research directions include scaling up our method to natural images using deep convolutional decoders.

Broader Impact Statement

This work proposes a general-purpose machine learning algorithm for representation learning which is trained using large amounts of unlabeled data. The representations learned by this algorithm reflect any biases present in the training data. Therefore, practitioners should apply techniques for identifying and mitigating the biases in the data when applicable. This notice is doubly important since the learned representations can be incorporated in numerous real-world applications such as medical diagnosis, surveillance systems, autonomous driving, and so on. These applications can have both positive and negative effect on society, and warrant extensive testing before being deployed in order to prevent harm or malicious use.

References

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Bo Chen, Gungor Polatkan, Guillermo Sapiro, David Blei, David Dunson, and Lawrence Carin. Deep learning with hierarchical convolutional factor analysis. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1887–1901, 2013.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745, 2006.

Basura Fernando, Hakan Bilen, Efstratios Gavves, and Stephen Gould. Self-supervised video representation learning with odd-one-out networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3636–3645, 2017.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 399–406, 2010.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21271–21284. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf`.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pp. 2146–2153. IEEE, 2009.

Koray Kavukcuoglu, Pierre Sermanet, Y lan Boureau, Karol Gregor, Michael Mathieu, and Yann L. Cun. Learning convolutional feature hierarchies for visual recognition. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems 23*, pp. 1090–1098. Curran Associates, Inc., 2010. URL `http://papers.nips.cc/paper/4133-learning-convolutional-feature-hierarchies-for-visual-recognition.pdf`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. URL `http://yann.lecun.com/exdb/mnist/`, 2010.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 498–515. Springer, 2015.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pp. 609–616, 2009.

Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse representation for color image restoration. *IEEE Transactions on image processing*, 17(1):53–69, 2007.

Julien Mairal, Francis R. Bach, and Jean Ponce. Sparse modeling for image and vision processing. *CoRR*, abs/1411.3230, 2014. URL `http://arxiv.org/abs/1411.3230`.

Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pp. 527–544. Springer, 2016.

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.

Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann L Cun. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pp. 1137–1144, 2007.

Snigdha Tariyal, Angshul Majumdar, Richa Singh, and Mayank Vatsa. Deep dictionary learning. *IEEE Access*, 4:10096–10109, 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

Kailun Wu, Yiwen Guo, Ziang Li, and Changshui Zhang. Sparse coding with gated learned ista. In *International Conference on Learning Representations*, 2019.

Takashi Yoshida and Kenichi Ohki. Natural images are reliably represented by sparse and variable populations of neurons in visual cortex. *Nature communications*, 11(1):1–19, 2020.

Kai Yu. Tutorial on deep learning: Sparse coding. *CVPR*, 2012.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*, 2021.

Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pp. 2528–2535. IEEE, 2010.

Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pp. 2018–2025. IEEE, 2011.

Zheng Zhang, Y. Xu, Jian Yang, Xuelong Li, and David Zhang. A survey of sparse representation: Algorithms and applications. *IEEE Access*, 3:490–530, 2015.

Joey Tianyi Zhou, K. Di, J. Du, Xi Peng, H. Yang, Sinno Jialin Pan, I. Tsang, Yong Liu, Z. Qin, and R. Goh. Sc2net: Sparse lstms for sparse coding. In *AAAI*, 2018.

A Appendix

A.1 Notation

Table 2 contains descriptions of the notation and symbols used in this work.

A.2 Training Details

All hyperparameter values are selected through grid search. For the constant step size η_k in FISTA, we consider values in the range of 0.01 to 100. In VDL experiments, we consider values for the hinge regularization coefficient β between 1e$-$1 and 100. In all our experiments, we use Adam (Kingma & Ba, 2014) as an optimizer for both the encoder $\mathcal{E}$ and decoder $\mathcal{D}$ with a batch size of 250. We use $L = 3$ iterations in the LISTA encoder (see Algorithm 1). We run the FISTA algorithm for a maximum of 200 iterations. The convergence criterion we set is: $\frac{\|\boldsymbol{z}^{(k)} - \boldsymbol{z}^{(k-1)}\|_2}{\|\boldsymbol{z}^{(k-1)}\|_2} < 1\text{e}{-}3$. FISTA's output is $\boldsymbol{z}^* = \boldsymbol{z}^{(k^*)}$ where k^* is the index of the first iteration for which the convergence criterion holds. Table 3 contains the hyperparameter values we used in all our experiments.

A.3 Variance Regularization Term

Figure 10 visualizes the variance regularization term $h(\boldsymbol{x}) = \left[\left(1 - \sqrt{\text{Var}(\boldsymbol{x})}\right)_+\right]^2$ for $\boldsymbol{x} \in \mathbb{R}^2$ in 3D.

A.3.1 Gradient Derivation

We compute the gradient of the hinge term in (8) with respect to one latent code's component $Z_{s,t}$:

$$\frac{\partial}{\partial Z_{s,t}} \sum_{j=1}^{l} \beta\Big[\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{j,:})}\Big)_+\Big]^2 \tag{14}$$

As a reminder of our notation, l is the latent dimension, n is the batch size, $\boldsymbol{Z} \in \mathbb{R}^{l \times n}$ stores the codes for all elements in a batch, and $\boldsymbol{Z}_{j,:} \in \mathbb{R}^n$ stores code values for the j-th latent component.

Note that:

$$\text{Var}(\boldsymbol{Z}_{j,:}) = \frac{1}{n-1} \sum_{i=1}^{n} (Z_{j,i} - \mu_j)^2, \tag{15}$$

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} Z_{j,i}. \tag{16}$$

Table 2: Descriptions of symbols and notation used.

Notation	Description
$\mathcal{E}$	LISTA-inspired encoder.
$\mathcal{D}$	Decoder: linear or fully-connected with one hidden layer.
m	Dimension of the decoder's hidden layer.
λ	Hyperparameter for the sparsity regularization.
β	Hyperparameter for the variance regularization of the latent codes.
γ	Hyperparameter encouraging the FISTA codes to be close to the encoder's predictions.
d	Dimension of the input data.
$\boldsymbol{y}$	Input sample in $\mathbb{R}^d$.
N	Total number of training samples.
n	Mini-batch size.
T	Number of ISTA/FISTA iterations.
η_k	Step size for ISTA/FISTA gradient step.
$\eta_{\mathcal{D}}$	Learning rate for the decoder.
$\eta_{\mathcal{E}}$	Learning rate for the encoder.
τ_α	Shrinkage function $[\tau_\alpha(\boldsymbol{x})]_j = \text{sign}(x_j)(\lvert x_j\rvert - \alpha)$.
$\tilde{\tau}_\alpha$	Non-negative shrinkage function $[\tilde{\tau}_\alpha(\boldsymbol{x})]_j = ([\tau_\alpha(\boldsymbol{x})]_j)_+$.
l	Dimension of the latent code.
$\boldsymbol{z}$	Latent code in $\mathbb{R}^l$.
$\boldsymbol{Z}$	Mini-batch of latent codes in $\mathbb{R}^{l\times n}$.
$\boldsymbol{W}$	Parametrization in $\mathbb{R}^{d\times l}$ of a linear dictionary $\mathcal{D}$.
$\boldsymbol{W}_1$	Parametrization in $\mathbb{R}^{m\times l}$ of the bottom layer of a fully-connected decoder $\mathcal{D}$ with one hidden layer.
$\boldsymbol{W}_2$	Parametrization of a the top layer of a fully-connected decoder $\mathcal{D}$ with one hidden layer in $\mathbb{R}^{d\times m}$.
$\boldsymbol{b}_1$	Parametrization of the bias term in $\mathbb{R}^m$ following the bottom layer of a fully-connected decoder $\mathcal{D}$ with one hidden layer.
$\boldsymbol{b}$	Parametrization of the bias term in the LISTA encoder (see 3.1).
SDL	Standard Dictionary Learning with a linear decoder whose columns have a fixed l_2 norm.
SDL-NL	Standard Dictionary learning with a non-linear fully-connected decoder. Each layer in the decoder has columns with a fixed l_2 norm.
VDL	Variance-regularized Dictionary Learning with a linear decoder in which regularization is applied to the sparse codes encouraging the variance across the latent components to be above a fixed threshold.
VDL-NL	Variance-regularized Dictionary Learning (as above) with a non-linear decoder.

The gradient in (14) is non-zero only for $j = s$ and $\sqrt{\text{Var}(\boldsymbol{Z}_{s,:})} < T$. Thus,

$$\frac{\partial}{\partial Z_{s,t}} \sum_{j=1}^{l} \beta\Big[\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{j,:})}\Big)_+\Big]^2 = \frac{\partial}{\partial Z_{s,t}} \beta\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}\Big)^2 \tag{17}$$

$$= 2\beta\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}\Big) \frac{\partial}{\partial Z_{s,t}} \Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}\Big) \tag{18}$$

$$= -2\beta\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}\Big) \frac{\partial}{\partial Z_{s,t}} \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})} \tag{19}$$

$$= -\beta \frac{\Big(T - \sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}\Big)}{\sqrt{\text{Var}(\boldsymbol{Z}_{s,:})}} \frac{\partial}{\partial Z_{s,t}} \text{Var}(\boldsymbol{Z}_{s,:}) \tag{20}$$

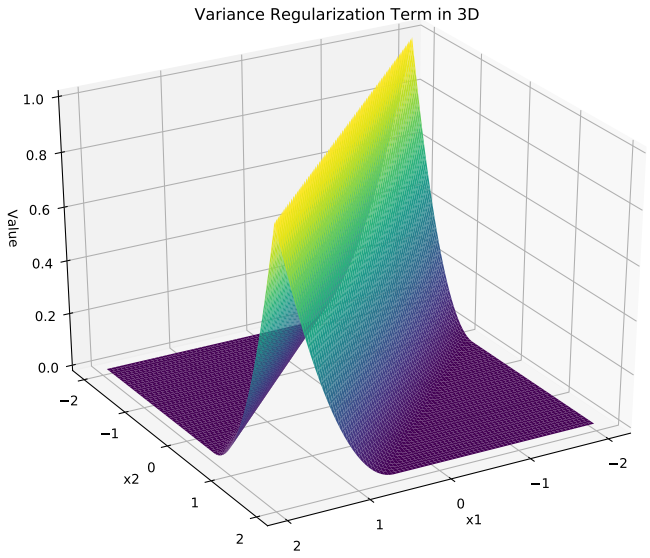


Figure 10: Visualizing the variance regularization function $h(\boldsymbol{x}) = \left[\left(1 - \sqrt{\mathrm{Var}(\boldsymbol{x})}\right)_+\right]^2$ for $\boldsymbol{x} \in \mathbb{R}^2$ in 3D.

Now, continuing with:

$$\frac{\partial}{\partial Z_{s,t}}\mathrm{Var}(\boldsymbol{Z}_{s,:}) = \frac{\partial}{\partial Z_{s,t}}\Big[\frac{1}{n-1}\sum_{i=1}^{n}(Z_{s,i}-\mu_s)^2\Big] \tag{21}$$

$$= \frac{1}{n-1}\Big[\frac{\partial}{\partial Z_{s,t}}(Z_{s,t}-\mu_s)^2 + \sum_{i\neq t}\frac{\partial}{\partial Z_{s,t}}(Z_{s,i}-\mu_s)^2\Big] \tag{22}$$

$$= \frac{1}{n-1}\Big[2(Z_{s,t}-\mu_s)\frac{\partial}{\partial Z_{s,t}}(Z_{s,t}-\mu_s) + \sum_{i\neq t}2(Z_{s,i}-\mu_s)\frac{\partial}{\partial Z_{s,t}}(Z_{s,i}-\mu_s)\Big] \tag{23}$$

$$= \frac{2}{n-1}\Big[(Z_{s,t}-\mu_s)\frac{\partial}{\partial Z_{s,t}}\Big(Z_{s,t}-\frac{1}{n}\sum_{m=1}^{n}Z_{s,m}\Big) + \sum_{i\neq t}(Z_{s,i}-\mu_s)\frac{\partial}{\partial Z_{s,t}}\Big(Z_{s,i}-\frac{1}{n}\sum_{m=1}^{n}Z_{s,m}\Big)\Big] \tag{24}$$

$$= \frac{2}{n-1}\Big[(Z_{s,t}-\mu_s)\Big(1-\frac{1}{n}\Big) + \sum_{i\neq t}(Z_{s,i}-\mu_s)\Big(-\frac{1}{n}\Big)\Big] \tag{25}$$

$$= \frac{2}{n-1}\Big[(Z_{s,t}-\mu_s) - \frac{1}{n}\sum_{i=1}^{n}(Z_{s,i}-\mu_s)\Big] \tag{26}$$

$$= \frac{2}{n-1}\Big[(Z_{s,t}-\mu_s) - \frac{1}{n}\sum_{i=1}^{n}Z_{s,i} + \frac{1}{n}n\mu_s\Big] \tag{27}$$

$$= \frac{2}{n-1}\Big[(Z_{s,t}-\mu_s) - \mu_s + \mu_s\Big] \tag{28}$$

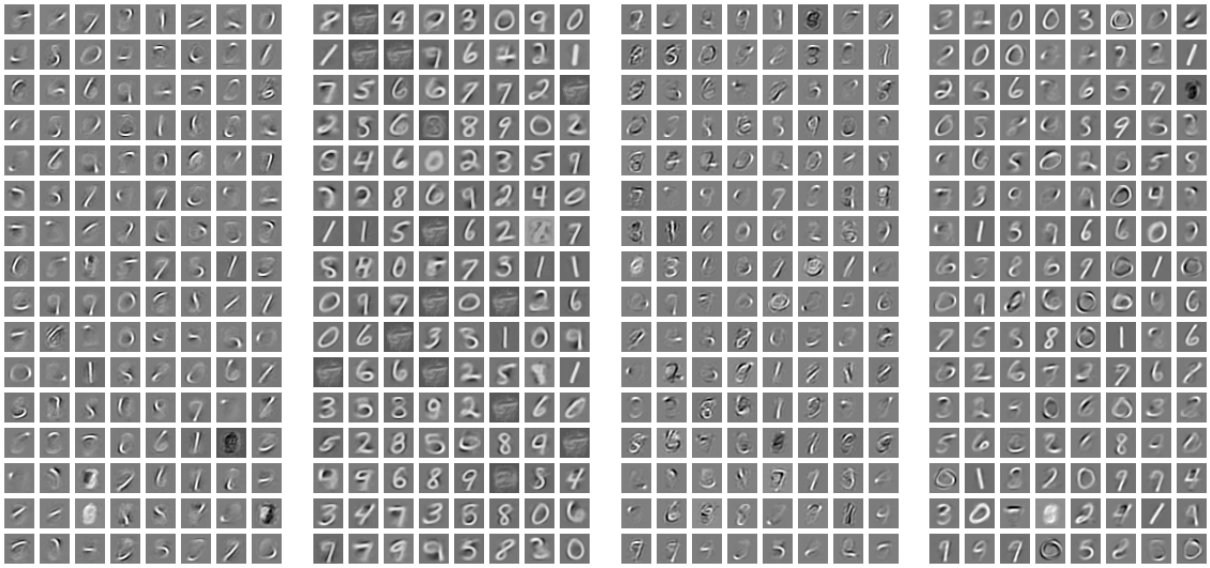
$$= \frac{2}{n-1}(Z_{s,t}-\mu_s), \tag{29}$$

and using the result in (20) we conclude that:

$$\frac{\partial}{\partial Z_{s,t}} \sum_{j=1}^{l} \beta \Big[\Big(T - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{j,:})} \Big)_+ \Big]^2 = \begin{cases} -\frac{2\beta}{n-1} \frac{\left(T - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})} \right)}{\sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})}} (Z_{s,t} - \mu_s) & \text{if } \sqrt{\mathrm{Var}(\boldsymbol{Z}_{s,:})} < T \\ 0 & \text{otherwise.} \end{cases} \tag{30}$$

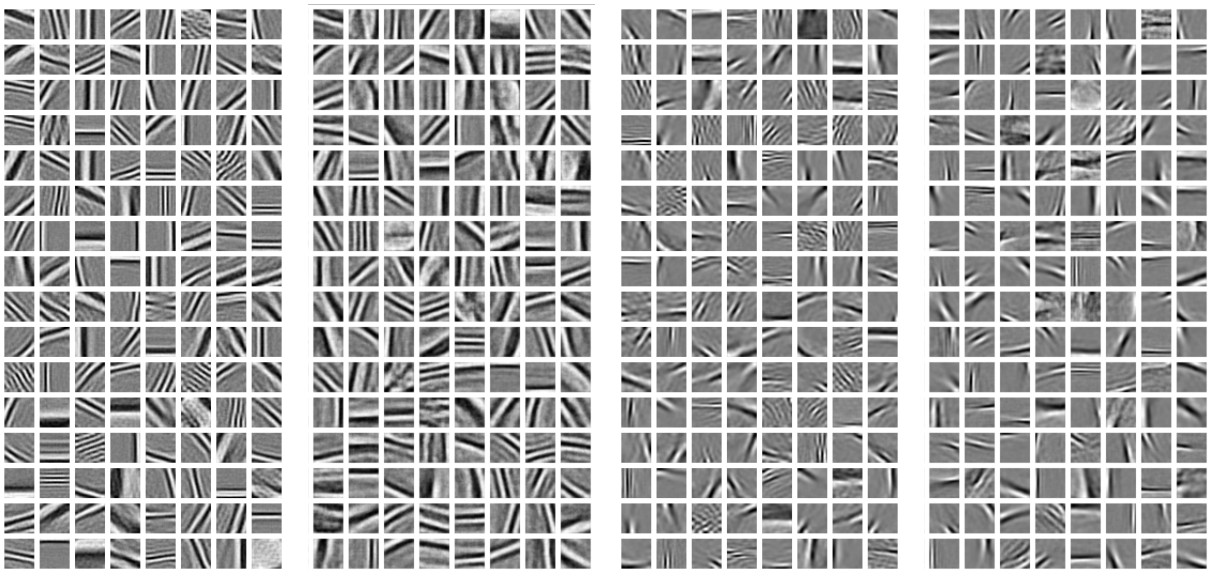
B Additional Visualizations

Figures 11a and 11c display 128 of the dictionary elements for the same SDL and VDL models as in Figures 2 and 3.



(a) SDL, $\lambda = 0.001$ (b) SDL, $\lambda = 0.005$ (c) VDL, $\lambda = 0.005$ (d) VDL, $\lambda = 0.02$

Figure 11: Dictionary elements for linear dictionaries with latent dimension $l = 128$ and different levels of sparsity regularization λ. SDL stands for dictionary learning in which the l_2 norm of the decoder's columns is fixed to 1. VDL stands for dictionary learning in which norms of the decoder's columns are not explicitly bounded but our proposed variance regularization on the latent codes is used. The SDL models in 2a and 2b produce reconstructions with average PSNR of 21.1 and 18.6 for codes with average sparsity level of 63% and 83% on the test set, respectively. The VDL models in 2c and 2d produce reconstructions with average PSNR of 20.7 and 17.7 for codes with average sparsity level of 69% and 91.8% on the test set, respectively.



(a) SDL, $\lambda = 0.002$ (b) SDL, $\lambda = 0.005$ (c) VDL, $\lambda = 0.005$ (d) VDL, $\lambda = 0.01$

Figure 12: Dictionary elements which resemble Gabor filters for SDL and VDL models with latent dimension $l = 256$ trained on ImageNet patches with different levels of sparsity regularization λ. The SDL models in 3a and 3b produce reconstructions with average PSNR of 26.9 and 24.0 and average sparsity level in the codes of 74.6% and 90.5% on the test set, respectively. The VDL models in Figures 3c and 3d produce reconstructions with average PSNR of 27.3 and 25.3 and average sparsity level in the codes of 77.3% and 89.2% on the test set, respectively.

dataset	model	ep	λ	γ	β	T	$\eta_{\mathcal{D}}$	$\eta_{\mathcal{E}}$	wd($\boldsymbol{b}_1$)	wd($\boldsymbol{b}$)	η_k
MNIST	SDL	200	0	1	0	-	1e−3	3e−4	-	0	1
MNIST	SDL	200	1e−4	1	0	-	1e−3	3e−4	-	0	1
MNIST	SDL	200	5e−4	1	0	-	1e−3	3e−4	-	0	1
MNIST	SDL	200	1e−3	1	0	-	1e−3	3e−4	-	0	1
MNIST	SDL	200	3e−3	1	0	-	1e−3	3e−4	-	0	1
MNIST	SDL	200	5e−3	1	0	-	1e−3	3e−4	-	0	1
MNIST	VDL	200	0	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	VDL	200	1e−3	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	VDL	200	3e−3	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	VDL	200	5e−3	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	VDL	200	1e−2	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	VDL	200	2e−2	5	10	0.5	3e−4	1e−4	-	0	0.5
MNIST	SDL-NL	200	0	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	SDL-NL	200	1e−3	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	SDL-NL	200	3e−3	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	SDL-NL	200	5e−3	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	SDL-NL	200	1e−2	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	SDL-NL	200	2e−2	1	0	-	1e−3	1e−4	1e−3	0	1
MNIST	VDL-NL	200	0	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
MNIST	VDL-NL	200	1e−3	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
MNIST	VDL-NL	200	3e−3	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
MNIST	VDL-NL	200	5e−3	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
MNIST	VDL-NL	200	1e−2	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
MNIST	VDL-NL	200	2e−2	100	10	0.5	3e−4*	1e−4	1e−3	0	0.5
ImageNet	SDL	100	0	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	SDL	100	5e−4	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	SDL	100	1e−3	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	SDL	100	2e−3	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	SDL	100	3e−3	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	SDL	100	5e−3	1	0	-	1e−3	1e−4	-	1e−2	0.5
ImageNet	VDL	100	0	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	VDL	100	1e−3	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	VDL	100	3e−3	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	VDL	100	5e−3	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	VDL	100	1e−2	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	VDL	100	1.5e−2	5	10	0.5	3e−4	1e−4	-	1e−2	0.5
ImageNet	SDL-NL	100	0	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	SDL-NL	100	1e−3	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	SDL-NL	100	3e−3	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	SDL-NL	100	5e−3	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	SDL-NL	100	8e−3	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	SDL-NL	100	1e−2	1	0	-	1e−3	1e−4	1e−2	1e−2	0.5
ImageNet	VDL-NL	100	0	20	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	1e−3	20	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	2e−3	20	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	3e−3	20	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	5e−3	20	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	1e−2	40	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5
ImageNet	VDL-NL	100	2e−2	40	10	0.5	5e−5*	1e−4	1e−1	1e−2	0.5

Table 3: Training hyperparameters. SDL indicates models in which columns of the decoder's layers have a fixed norm of 1. VDL indicates models in which variance regularization is applied to the codes during inference. (*) means that the learning rate for the decoder $\eta_{\mathcal{D}}$ is annealed by half every 30 epochs. wd stands for weight decay.