Unsupervised Feature Learning for Computer Vision

by

Rostislav Goroshin

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2015

Professor Yann LeCun

Dedication

Parents and friends

Acknowledgements

Abstract

Much of computer vision has been devoted to the question of representation through feature extraction. Ideal features transform raw pixel intensity values to a representation in which common problems such as object identification, tracking, and segmentation are easier to solve. Recently, deep feature hierarchies have proven to be immensely successful at solving many problems in computer vision. In the supervised setting, these hierarchies are trained to solve specific problems by minimizing an objective function of the data and problem specific label information. Recent findings suggest that despite being trained on a specific task, the learned features can be transferred across multiple visual tasks. These findings suggests that there exists a generically useful feature representation for natural visual data.

This work aims to uncover the principles that lead to these generic feature representations in the unsupervised setting, which does not require problem specific label information. We begin by reviewing relevant prior work, particularly the literature on autoencoder networks and energy based learning. We introduce a new regularizer for autoencoders that plays an analogous role to the partition function in probabilistic graphical models. Next we explore the role of specialized encoder architectures for sparse inference. The remainder of the thesis explores visual feature learning from video. We establish a connection between slow-feature learning and metric learning, and experimentally demonstrate that semantically coherent metrics can be learned from natural videos. Finally, we posit that useful features linearize natural image transformations in video. To this end, we introduce a new architecture and loss for training deep feature hierarchies that linearize the transformations observed in unlabeled natural video sequences by learning to predict future frames in the presence of uncertainty.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Vision is the ability to make inferences about the natural world from the patterns of light captured by the eyes of living organism or cameras in machines. A digital image is a discrete two dimensional array of intensity values referred to as pixels. Though each pixel corresponds to a potentially independent light intensity measurement, these intesities exhibit strong spatial correlations in natural images [44]. However, this structure is not explicitly captured by the camera sensor; each pixel is stored as a separate light intensity measurement in memory. Therefore it is natural to think of raw images as points in high dimensional space in which each pixel corresponds to an independant variable. Because of this structure, or dependency between adjacent pixels, the number of possible *natural* images is far smaller than the number of possible images. This observation has a geometric interpretation: the *intrinsic* dimension of natural images is much lower than the ambient dimension, thus natural images are concentrated around a low-dimensional “manifold” [3, 45, 43]. Similarly, videos can be imagined as one dimensional trajectories on this manifold parametrized by time [16]. Figure 1.1 is an illustrative visualization of the distribution of random images (left) and natural images

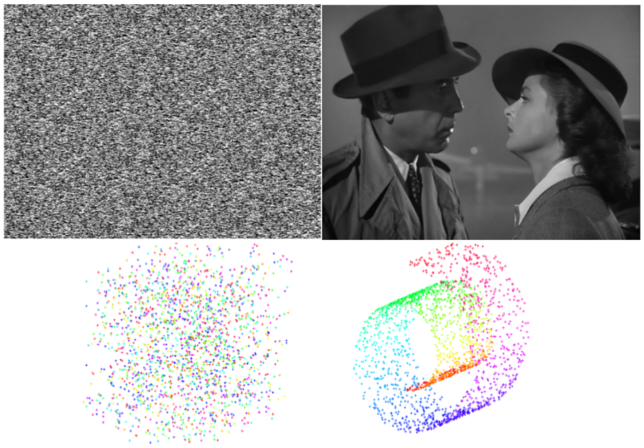


Figure 1.1: Top left: random image, Top right: natural image, Bottom left: visualization of unstructured data, Bottom right: visualization of data with intrinsic structure

(right).

The role of visual features is to transform the raw input image to a representation that facilitates inferences about the natural world by modeling commonly occurring dependencies between input pixels. This can be restated from the manifold perspective: visual features model the natural data manifold. This so called "manifold theory of natural data" allows for a more geometrically intuitive interpretation of problems in computer vision. Indeed, many feature learning algorithms can be related or even derived using the manifold theory perspective of natural data.

Historically, approaches for constructing visual features in computer vision have been broadly divided into two categories: "hand-crafted" features are designed by a human expert, whereas "learned" features use techniques from machine learning to learn the features automatically from the problem related data itself. Both types of features are often optimized in order to solve specific problems. Examples of hand crafted features (also called "descriptors") include the Scale Invariant Feature Transform (SIFT) [31], Histogram of Gradients (HoG) [10], and "gist" [33] features. Though these hand

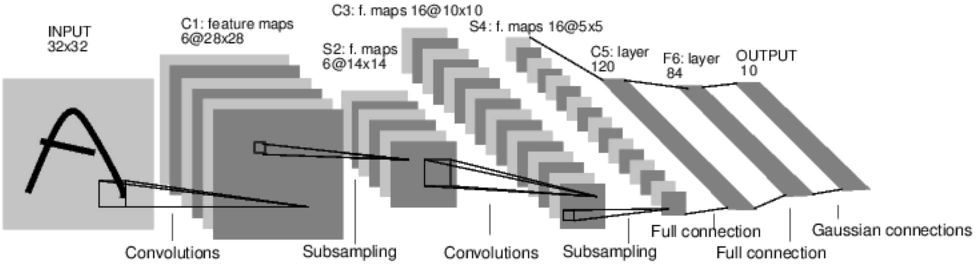


Figure 1.2: The LeNet5 network was trained to recognize hand written characters in order to automatically process bank checks

crafted features paved the way for many breakthroughs in computer vision, recent breakthroughs have largely been attributed to feature learning with deep convolutional networks (ConvNets) [13, 29, 28]. Remarkably, these hand-crafted features bear striking similarity to shallow ConvNets. The main differences between ConvNets and these features are that: (i) ConvNet features are optimized or “learned” for specific problems, and (ii) they are arranged and learned in a hierarchical fashion. Traditional ConvNet layers are composed of trainable convolutional filter banks, interspersed with point-wise nonlinearities and subsampling (or “pooling”) layers. A now famous ConvNet architecture, dubbed LeNet5 [29] is shown in Figure 1.2.

Interestingly, the hierarchical representations of ConvNets were inspired by studies of natural visual systems. A long standing belief in neuroscience is that the visual cortex also has a hierarchical organization [21, 12]. Simply put, there is a sequence of processing steps which eventually results in our visual perception of the world. Recent work has revealed strong empirical similarities between the artificial feature hierarchies of ConvNets and natural visual hierarchies of the animal visual system [50]. Though the hierarchical representations learned by ConvNets are not completely understood, it is clear that the sequence of operations corresponding to each layer transforms the data in a way that makes the task easier to solve for the final layer, often a linear operator.

Recent work suggests that early stages of the hierarchy learned by ConvNets on specific *supervised learning* tasks such as classification, can be transfered to other tasks, such as segmentation [51]. This finding suggests that these representations are not task specific but are generically useful. The goal of this thesis is to explore *unsupervised learning* principles which lead to generically useful image representations.

Chapter 2

Related Work

Chapter 3

Saturating Auto-Encoders

3.1 Introduction

An auto-encoder is a conceptually simple neural network used for obtaining useful data representations through unsupervised training. It is composed of an encoder which outputs a hidden (or latent) representation and a decoder which attempts to reconstruct the input using the hidden representation as its input. Training consists of minimizing a reconstruction cost such as L_2 error. However this cost is merely a proxy for the true objective: to obtain a useful latent representation. Auto-encoders can implement many dimensionality reduction techniques such as PCA and Sparse Coding (SC) [11] [35] [18]. This makes the study of auto-encoders very appealing from a theoretical standpoint. In recent years, renewed interest in auto-encoders networks has mainly been due to their empirical success in unsupervised feature learning [37][40][42][46].

When minimizing only reconstruction cost, the standard auto-encoder does not typically learn any meaningful hidden representation of the data. Well known theoretical and experimental results show that a linear auto-encoder with trainable encoding and decoding matrices, W^e and W^d respectively, learns the identity function if W^e and W^d are full rank or over-complete. The linear auto-encoder learns the principle variance directions (PCA) if W^e and W^d are rank deficient [11]. It has been observed that other representations can be obtained by regularizing the latent representation. This approach is exemplified by the Contractive and Sparse Auto-Encoders [42] [37] [40]. Intuitively, an auto-encoder with limited capacity will focus its resources on reconstructing portions of the input space in which data samples occur most frequently. From an energy based perspective, auto-encoders achieve low reconstruction cost in portions of the input space with high data density (recently, [1] has examined this perspective in depth). If the data occupies some low dimensional manifold in the higher dimensional input space then minimizing reconstruction error achieves low energy on this manifold. Useful latent state regularizers raise the energy of points that do not lie on the manifold, thus playing an analogous role to minimizing the partition function in maximum likelihood models. In this work we introduce a new type of regularizer that does this explicitly for auto-encoders with a non-linearity that contains at least one flat (zero gradient) region. We show examples where this regularizer and the choice of nonlinearity determine the feature set that is learned by the auto-encoder.

3.2 Latent State Regularization

Several auto-encoder variants which regularize their latent states have been proposed, they include the sparse auto-encoder and the contractive auto-encoder[37][40][42]. The sparse auto-encoder includes an over-complete basis in the encoder and imposes a sparsity inducing (usually L_1) penalty on the hidden activations.

This penalty prevents the auto-encoder from learning to reconstruct all possible points in the input space and focuses the expressive power of the auto-encoder on representing the data-manifold. Similarly, the contractive auto-encoder avoids trivial solutions by introducing an auxiliary penalty which measures the square Frobenius norm of the Jacobian of the latent representation with respect to the inputs. This encourages a constant latent representation except around training samples where it is counteracted by the reconstruction term. It has been noted in [42] that these two approaches are strongly related. The contractive auto-encoder explicitly encourages small entries in the Jacobian, whereas the sparse auto-encoder is encouraged to produce mostly zero (sparse) activations which can be designed to correspond to mostly flat regions of the nonlinearity, thus also yielding small entries in the Jacobian.

3.2.1 Saturating Auto-Encoder through Complementary Nonlinearities

Our goal is to introduce a simple new regularizer which explicitly raises reconstruction error for inputs not near the data manifold. Consider activation functions with at least one flat region; these include shrink, rectified linear, and saturated linear (Figure 3.1). Auto-encoders with such nonlinearities lose their ability to accurately reconstruct inputs which produce activations in the zero-gradient regions of their activation functions. Let us denote the auto-encoding function $x_r = G(x, W)$, x being the input, W the trainable parameters in the auto-encoder, and x_r the reconstruction. One can define an energy surface through the reconstruction error:

$$E_W(x) = ||x - G(x, W)||^2$$

Let's imagine that G has been trained to produce a low reconstruction error at a particular data point x^*. If G is constant when x varies along a particular direction v, then the energy will grow quadratically along that particular direction as x moves away from x^*. If G is trained to produce low reconstruction errors on a set of samples while being subject to a regularizer that tries to make it constant in as many directions as possible, then the reconstruction energy will act as a *contrast function* that will take low values around areas of high data density and larger values everywhere else (similarly to a negative log likelihood function for a density estimator).

The proposed auto-encoder is a simple implementation of this idea. Using the notation $W = \{W^e, B^e, W^d, B^d\}$, the auto-encoder function is defined as

$$G(x, W) = W^d F(W^e x + B^e) + B^d$$

where W^e, B^e, W^d, and B^d are the encoding matrix, encoding bias, decoding matrix, and decoding bias, respectively, and F is the vector function that applies the scalar function f to each of its components. f will be designed to have "flat spots", i.e. regions where the derivative is zero (also referred to as the saturation region).

The loss function minimized by training is the sum of the reconstruction energy $E_W(x) = ||x - G(x, W)||^2$ and a term that pushes the components of $W^e x + B^e$ towards the flat spots of f. This is performed through the use of a *complementary function* f_c, associated with the non-linearity $f(z)$. The basic idea is to design $f_c(z)$ so that its value corresponds to the distance of z to one of the flat spots of $f(z)$. Minimizing $f_c(z)$ will push z towards the flat spots of $f(z)$. With this in mind, we introduce a penalty of the form $f_c(\sum_{j=1}^{d} W_{ij}^e x_j + b_i^e)$ which encourages the argument to be in the saturation regime of the activation function (f). We refer to auto-encoders which include this

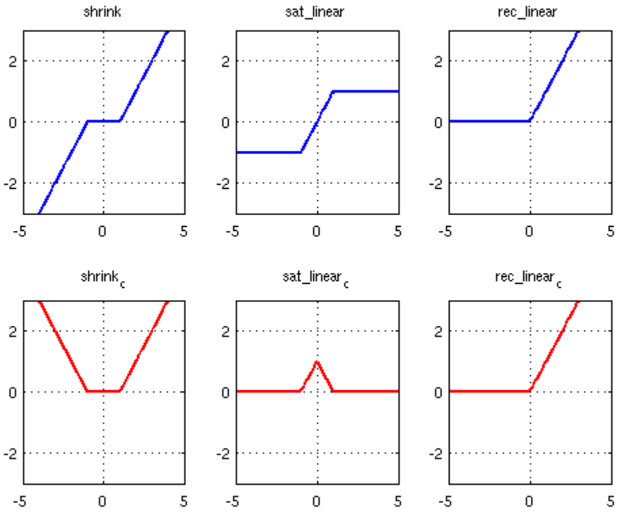


Figure 3.1: Three nonlinearities (top) with their associated complementary regularization functions(bottom).

regularizer as Saturating Auto-Encoders (SATAEs). For activation functions with zero-gradient regime(s) the complementary nonlinearity (f_c) can be defined as the distance to the nearest saturation region. Specifically, let $S = \{z \mid f'(z) = 0\}$ then we define $f_c(z)$ as:

$$f_c(z) = \inf_{z' \in S} |z - z'|. \tag{3.1}$$

Figure 1 shows three activation functions and their associated complementary nonlinearities. The complete loss to be minimized by a SATAE with nonlinearity f is:

$$L = \sum_{x \in D} \frac{1}{2} ||x - \left(W^d F(W^e x + B^e) + B^d\right)||^2 + \alpha \sum_{i=1}^{d_h} f_c(W_i^e x + b_i^e), \tag{3.2}$$

where d_h denotes the number of hidden units. The hyper-parameter α regulates the trade-off between reconstruction and saturation.

3.3 Effect of the Saturation Regularizer

We will examine the effect of the saturation regularizer on auto-encoders with a variety of activation functions. It will be shown that the choice of activation function is a significant factor in determining the type of basis the SATAE learns. First, we will present results on toy data in two dimensions followed by results on higher dimensional image data.

3.3.1 Visualizing the Energy Landscape

Given a trained auto-encoder the reconstruction error can be evaluated for a given input x. For low-dimensional spaces ($\mathbb{R}^n$, where $n \leq 3$) we can evaluate the reconstruction error on a regular grid in order to visualize the portions of the space which are well represented by the auto-encoder. More specifically we can compute $E(x) = \frac{1}{2}||x - x_r||^2$ for all x within some bounded region of the input space. Ideally, the reconstruction energy will be low for all x which are in the training set and high elsewhere. Figures 3.2 and 3.3 depict the resulting reconstruction energy for inputs $x \in \mathbb{R}^2$, and $-1 \leq x_i \leq 1$. Black corresponds to low reconstruction energy. The training data consists of a one dimensional manifold shown overlain in yellow. Figure 3.2 shows a toy example for a SATAE which uses ten basis vectors and a shrink activation function. Note that adding the saturation regularizer decreases the volume of the space which is well reconstructed, however good reconstruction is maintained on or near the training data manifold. The auto-encoder in Figure 3.3 contains two encoding basis vectors (red), two decoding basis vectors (green), and uses a saturated-linear activation function. The encoding and decoding bases are unconstrained. The unregularized auto-encoder learns an orthogonal basis with a random orientation. The region of the space which is well reconstructed

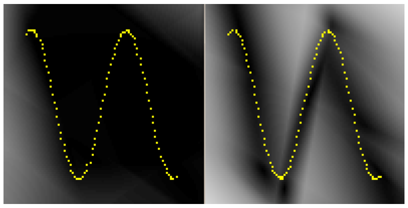


Figure 3.2: Energy surfaces for unregularized (left), and regularized (right) solutions obtained on SATAE-shrink and 10 basis vectors. Black corresponds to low reconstruction energy. Training points lie on a one-dimensional manifold shown in yellow.

corresponds to the outer product of the linear regions of two activation functions; beyond that the error increases quadratically with the distance. Including the saturation regularizer induces the auto-encoder basis to align with the data and to operate in the saturation regime at the extreme points of the training data, which limits the space which is well reconstructed. Note that because the encoding and decoding weights are separate and unrestricted, the encoding weights were scaled up to effectively reduce the width of the linear regime of the nonlinearity.

3.3.2 SATAE-shrink

Consider a SATAE with a shrink activation function and shrink parameter λ. The corresponding complementary nonlinearity, derived using Equation 1 is given by:

$$shrink_c(x) = \begin{cases} abs(x), & |x| > \lambda \\ 0, & \text{elsewhere} \end{cases} .$$

Note that $shrink_c(W^e x + b^e) = abs(shrink(W^e x + b^e))$, which corresponds to an L_1 penalty on the activations. Thus this SATAE is equivalent to a sparse auto-encoder with a shrink activation function. Given the equivalence to the sparse auto-encoder we anticipate the same scale ambiguity which occurs with L_1 regularization. This ambiguity

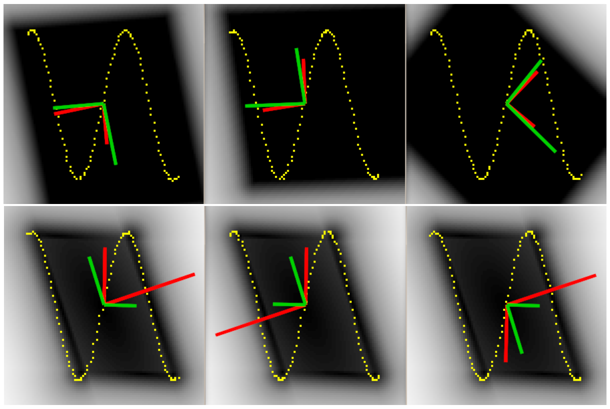


Figure 3.3: SATAE-SL toy example with two basis elements. Top Row: three randomly initialized solutions obtained with no regularization. Bottom Row: three randomly initialized solutions obtained with regularization.

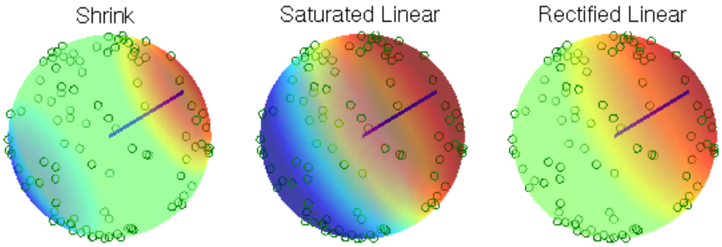


Figure 3.4: Geometric visualization of non-linearities

can be avoided by normalizing the decoder weights to unit norm. It is expected that the SATAE-shrink will learn similar features to those obtained with a sparse auto-encoder, and indeed this is what we observe. Figure 3.6(c) shows the decoder filters learned by an auto-encoder with shrink nonlinearity trained on gray-scale natural image patches. One can recognize the expected Gabor-like features when the saturation penalty is activated. When trained on the binary MNIST dataset the learned basis is comprised of portions of digits and strokes. Nearly identical results are obtained with a SATAE which uses a rectified-linear activation function. This is because a rectified-linear function with an encoding bias behaves as a positive only shrink function, similarly the complementary function is equivalent to a positive only L_1 penalty on the activations.

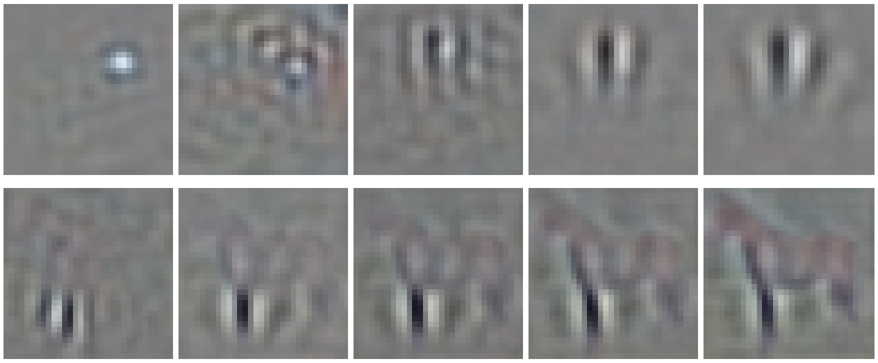


Figure 3.5: Evolution of two filters with increasing saturation regularization for a SATAE-SL trained on CIFAR-10. Filters corresponding to larger values of α were initialized using the filter corresponding to the previous α. The regularization parameter was varied from 0.1 to 0.5 (left to right) in the top five images and 0.5 to 1 in the bottom five

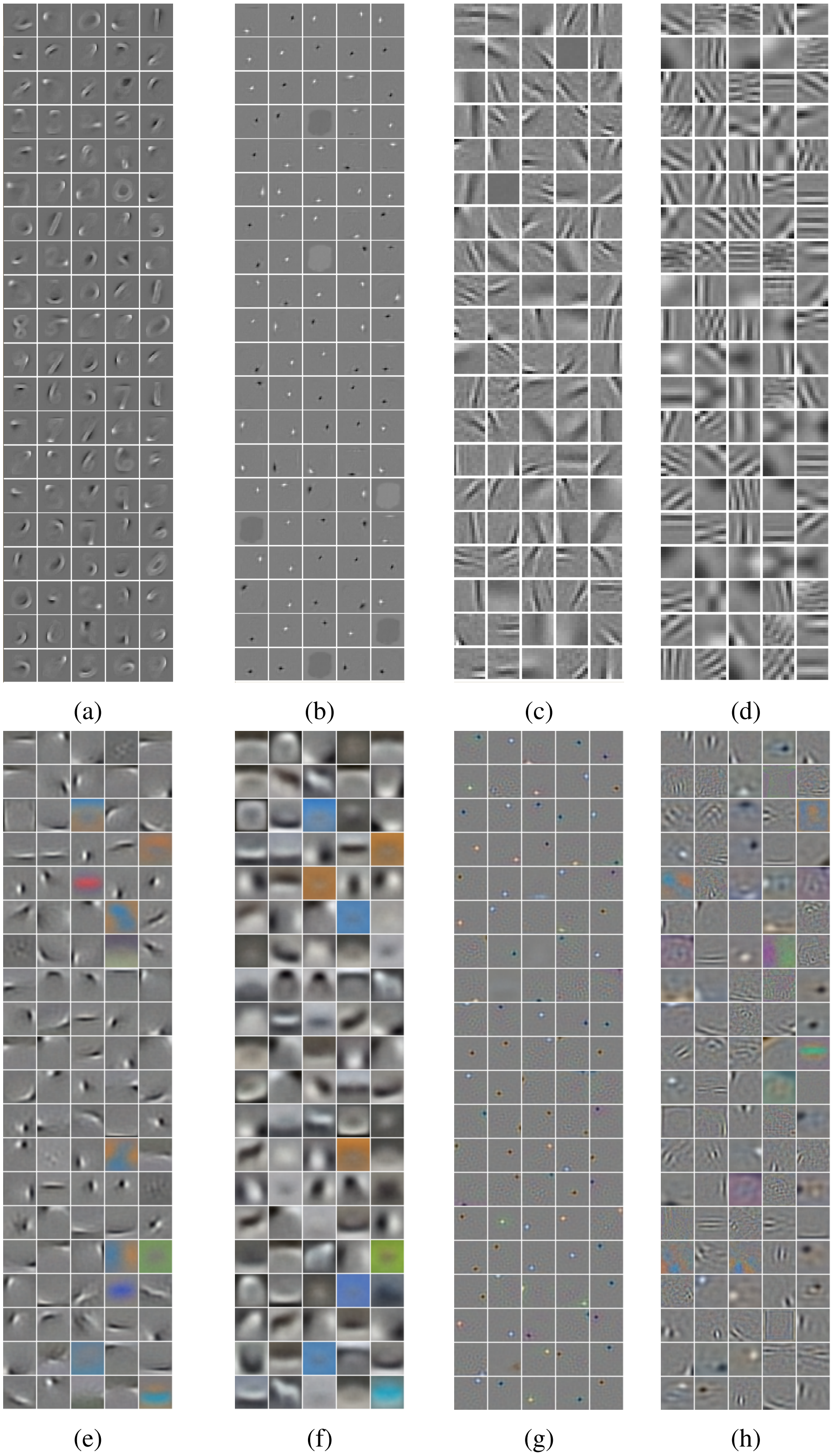


Figure 3.6: Basis elements learned by the SATAE using different nonlinearities on: 28x28 binary MNIST digits, 12x12 gray scale natural image patches, and CIFAR-10. (a) SATAE-shrink trained on MNIST, (b) SATAE-saturated-linear trained on MNIST, (c) SATAE-shrink trained on natural image patches, (d) SATAE-saturated-linear trained on natural image patches, (e)-(f) SATAE-shrink trained on CIFAR-10 with $\alpha = 0.1$ and $\alpha = 0.5$, respectively, (g)-(h) SATAE-SL trained on CIFAR-10 with $\alpha = 0.1$ and $\alpha = 0.6$, respectively.

3.3.3 SATAE-saturated-linear

Unlike the SATAE-shrink, which tries to compress the data by minimizing the number of active elements; the SATAE saturated-linear (SATAE-SL) tries to compress the data by encouraging the latent code to be as close to binary as possible. Without a saturation penalty this auto-encoder learns to encode small groups of neighboring pixels. More precisely, the auto-encoder learns the identity function on all datasets. An example of such a basis is shown in Figure 3.6(b). With this basis the auto-encoder can perfectly reconstruct any input by producing small activations which stay within the linear region of the nonlinearity. Introducing the saturation penalty does not have any effect when training on binary MNIST. This is because the scaled identity basis is a global minimizer of Equation 2 for the SATAE-SL on any binary dataset. Such a basis can perfectly reconstruct any binary input while operating exclusively in the saturated regions of the activation function, thus incurring no saturation penalty. On the other hand, introducing the saturation penalty when training on natural image patches induces the SATAE-SL to learn a more varied basis (Figure 3.6(d)).

3.3.4 Experiments on CIFAR-10

SATAE auto-encoders with 100 and 300 basis elements were trained on the CIFAR-10 dataset, which contains small color images of objects from ten categories. In all of our experiments the auto-encoders were trained by progressively increasing the saturation penalty (details are provided in the next section). This allowed us to visually track the effect of the saturation penalty on individual basis elements. Figure 3.6(e)-(f) shows the basis learned by SATAE-shrink with small and large saturation penalty, respectively. Increasing the saturation penalty has the expected effect of reducing the number of nonzero activations. As the saturation penalty increases, active basis elements become responsible for reconstructing a larger portion of the input. This induces the basis elements to become less spatially localized. This effect can be seen by comparing corresponding filters in Figure 3.6(e) and (f). Figures 3.6(g)-(h) show the basis elements learned by SATAE-SL with small and large saturation penalty, respectively. The basis learned by SATAE-SL with a small saturation penalty resembles the identity basis, as expected (see previous subsection). Once the saturation penalty is increased small activations become more heavily penalized. To increase their activations the encoding basis elements may increase in magnitude or align themselves with the input. However, if the encoding and decoding weights are tied (or fixed in magnitude) then reconstruction error would increase if the weights were merely scaled up. Thus the basis elements are forced to align with the data in a way that also facilitates reconstruction. This effect is illustrated in Figure 3.5 where filters corresponding to progressively larger values of the regularization parameter are shown. The top half of the figure shows how an element from the identity basis ($\alpha = 0.1$) transforms to a localized edge ($\alpha = 0.5$). The bottom half of the figure shows how a localized edge ($\alpha = 0.5$) progressively transforms to a template of a horse ($\alpha = 1$).

3.4 Experimental Details

Because the regularizer explicitly encourages activations in the zero gradient regime of the nonlinearity, many encoder basis elements would not be updated via back-propagation through the nonlinearity if the saturation penalty were large. In order to allow the basis elements to deviate from their initial random states we found it necessary to progressively increase the saturation penalty. In our experiments the weights obtained at a minimum of Equation 2 for a smaller value of α were used to initialize the optimization for a larger value of α. Typically, the optimization began with $\alpha = 0$ and was progressively increased to $\alpha = 1$ in steps of 0.1. The auto-encoder was trained for 30 epochs at each value of α. This approach also allowed us to track the evolution of basis elements as a function of α (Figure 3.5). In all experiments data samples were normalized by subtracting the mean and dividing by the standard deviation of the dataset. The auto-encoders used to obtain the results shown in Figure 3.6 (a),(c)-(f) used 100 basis elements, others used 300 basis elements. Increasing the number of elements in the basis did not have a strong qualitative effect except to make the features represented by the basis more localized. The decoder basis elements of the SATAEs with shrink and rectified-linear nonlinearities were reprojected to the unit sphere after every 10 stochastic gradient updates. The SATAEs which used saturated-linear activation function were trained with tied weights. All results presented were obtained using stochastic gradient descent with a constant learning rate of 0.05.

3.5 Discussion

In this work we have introduced a general and conceptually simple latent state regularizer. It was demonstrated that a variety of feature sets can be obtained using a single framework. The utility of these features depend on the application. In this section we extend the definition of the saturation regularizer to include functions without a zero-gradient region. The relationship of SATAEs with other regularized auto-encoders will be discussed. We conclude with a discussion on future work.

3.5.1 Extension to Differentiable Functions

We would like to extend the saturation penalty definition (Equation 1) to differentiable functions without a zero-gradient region. An appealing first guess for the complimentary function is some positive function of the first derivative, $f_c(x) = |f'(x)|$ for instance. This may be an appropriate choice for monotonic activation functions which have their lowest gradient regions at the extrema (e.g. sigmoids). However some activation functions may contain regions of small or zero gradient which have negligible extent, at the extrema for instance. We would like our definition of the complimentary function to not only measure the local gradient in some region, but to also measure it's extent. For this purpose we employ the concept of average variation over a finite interval. We define the average variation of f at x in the positive and negative directions at scale l, respectively as:

$$\begin{aligned}
\Delta_l^+ f(x) &= \frac{1}{l}\int_x^{x+l} |f'(u)|du = |f'(x)| * \Pi_l^+(x) \\
\Delta_l^- f(x) &= \frac{1}{l}\int_{x-l}^{x} |f'(u)|du = |f'(x)| * \Pi_l^-(x).
\end{aligned}$$

Where $*$ denotes the continuous convolution operator. $\Pi_l^+(x)$ and $\Pi_l^-(x)$ are uniform averaging kernels in the positive and negative directions, respectively. Next, define a directional measure of variation of f by integrating the average variation at all scales.

$$\begin{aligned}
M^+ f(x) &= \int_0^{+\infty} \Delta_l^+ f(x) w(l) dl = \left[\int_0^{+\infty} w(l)\Pi_l^+(x) dl\right] * |f'(x)| \\
M^- f(x) &= \int_0^{+\infty} \Delta_l^- f(x) w(l) dl = \left[\int_0^{+\infty} w(l)\Pi_l^-(x) dl\right] * |f'(x)|.
\end{aligned}$$

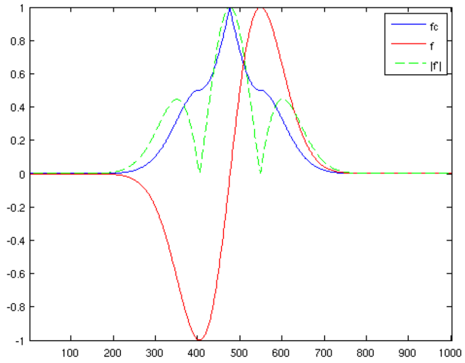


Figure 3.7: Illustration of the complimentary function (f_c) as defined by Equation 3 for a non-monotonic activation function (f). The absolute derivative of f is shown for comparison.

Where $w(l)$ is chosen to be a sufficiently fast decreasing function of l to insure convergence of the integral. The integral with which $|f'(x)|$ is convolved in the above equation evaluates to some decreasing function of x for Π^+ with support $x \geq 0$. Similarly, the integral involving Π^- evaluates to some increasing function of x with support $x \leq 0$. This function will depend on $w(l)$. The functions $M^+f(x)$ and $M^-f(x)$ measure the average variation of $f(x)$ at all scales l in the positive and negative direction, respectively. We define the complimentary function $f_c(x)$ as:

$$f_c(x) = min(M^+f(x), M^-f(x)). \tag{3.3}$$

An example of a complimentary function defined using the above formulation is shown in Figure 3.7. Whereas $|f'(x)|$ is minimized at the extrema of f, the complimentary function only plateaus at these locations.

3.5.2 Relationship with the Contractive Auto-Encoder

Let h_i be the output of the i^{th} hidden unit of a single-layer auto-encoder with pointwise nonlinearity $f(\cdot)$. The regularizer imposed by the contractive auto-encoder (CAE) can be expressed as follows:

$$\sum_{ij}\left(\frac{\partial h_i}{\partial x_j}\right)^2 = \sum_{i}^{d_h}\left(f'(\sum_{j=1}^{d} W_{ij}^e x_j + b_i)^2 \|W_i^e\|^2\right),$$

where x is a d-dimensional data vector, $f'(\cdot)$ is the derivative of $f(\cdot)$, b_i is the bias of the i^{th} encoding unit, and W_i^e denotes the i^{th} row of the encoding weight matrix. The first term in the above equation tries to adjust the weights so as to push the activations into the low gradient (saturation) regime of the nonlinearity, but is only defined for differentiable activation functions. Therefore the CAE indirectly encourages operation in the saturation regime. Computing the Jacobian, however, can be cumbersome for deep networks. Furthermore, the complexity of computing the Jacobian is $O(d \times d_h)$, although a more efficient implementation is possible [42], compared to the $O(d_h)$ for the saturation penalty.

3.5.3 Relationship with the Sparse Auto-Encoder

In Section 3.2 it was shown that SATAEs with shrink or rectified-linear activation functions are equivalent to a sparse auto-encoder. Interestingly, the fact that the saturation penalty happens to correspond to L_1 regularization in the case of SATAE-shrink agrees with the findings in [18]. In their efforts to find an architecture to approximate inference in sparse coding, Gregor et al. found that the shrink function is particularly compatible with L_1 minimization. Equivalence to sparsity only for some activation functions suggests that SATAEs are a generalization of sparse auto-encoders. Like the sparsity penalty, the saturation penalty can be applied at any point in a deep network for the same computational cost. However, unlike the sparsity penalty the saturation penalty is adapted to the nonlinearity of the particular layer to which it is applied.

Chapter 4

Convolutional Sparse Inference

Sparse coding inspired many early works in deep feature learning [35, 39, 40], as well as early works in unsupervised learning of convolutional feature hierarchies [25]. All of these works rely heavily on solving the sparse inference problem described in Chapter 2. However traditional iterative solvers can be computationally expensive and are not formulated as functional mappings that are amenable to network implementations [2]. The work by Gregor and LeCun dubbed "LISTA" [18] proposed a specialized network architecture for learning to predict sparse codes. However in that work LISTA networks were trained to directly predict the codes found using intirative inference algorithms. In this chapter we will empirically evaluate LISTA encoders under a wider variety of conditions. This will include training convolutional LISTA networks to learn sparse inference in convolutional dictionaries by directly minimizing the lasso loss.

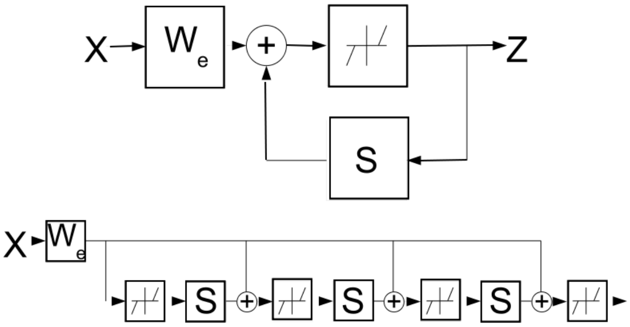


Figure 4.1: LISTA network architecture

4.1 Convolutional-LISTA

Recall the iterative ISTA algorithm for solving the relaxed sparse inference problem known as the lasso was discussed in Chapter **??**. The lasso loss is given by:

$$L_{lasso} = \frac{1}{2}||X - W_d Z|| + \alpha|Z|_1 \tag{4.1}$$

The LISTA network architecture is *derived* by expressing the ISTA algorithm as a recurrent network, and "unrolling" into a finite number of loops with shared-weight stages. The recurrent and unrolled networks (three-loops) are shown in Figure 4.1. Note that $S = I - \frac{1}{L} W_d^T W_d$, where W_d is the decoder and L is the upper bound of $W_d^T W_d$. Although the "shrinkage" nonlinearity is depicted in the architecture of Figure 4.1, we used rectified linear (ReLU) non-linearities which produce non-negative sparse codes. The above network can be made convolutional by replacing the linear operators W_e and S with convolutional filter banks. In order to be able to compute the reconstruction error in Equation 4.1, the convolutional synthesis operator must produce outputs of the same size as the input. This can be accomplished by using "same" convolutions or by cropping the input and computing the reconstruction error in the "valid" regions. As in ordinary convolutional networks, each convolutional layer produces multiple output planes. For example, if the encoder takes in images of 3-input planes and produces n-output planes then the S convolution stage takes in n-input planes and produces n-output planes. Convolutional dictionaries are massively overcomplete, making sparse inference a potentially much harder problem [25].

4.2 Learning to Perform Sparse Inference

Training networks to perform sparse inference was originally proposed in [24], the specialized LISTA architecture was proposed later in [18]. In [18] networks were trained to explicitly minimize the L^2 distance between the predicted and ground truth codes obtained via iterative sparse inference. This requires pre-training a dictionary and constructing a dataset consisting of sparse codes corresponding to minima of the lasso loss. However, in [24] the networks were implicitly trained to produce sparse codes by directly minimizing a variant of the lasso loss. The variant included an additional term which encouraged the predicted codes to be close to the codes which minimize the lasso loss. In the context of dictionary learning it is desirable to learn the encoder and decoder jointly, (i.e. training an auto-encoder) as opposed to learning a decoder first then training an encoder to predict the sparse codes. This section will present experimental results which empirically answer the following questions:

- Does the convolutional LISTA architecture outperform other architectures when minimizing the lasso loss with a fixed dictionary?
- Does the convolutional LISTA architecture outperform other architectures when minimizing the lasso loss with a learned dictionary?

The following experiments were performed on the whitened CIFAR-10 dataset (Figure 4.3), all reported results were obtained on the CIFAR-10 test set. Fixed-dictionary exper-

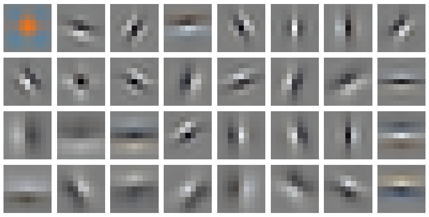


Figure 4.2: Pre-trained decoder used for fixed-decoder experiments

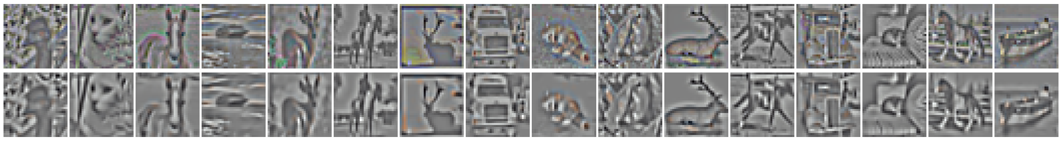


Figure 4.3: Whitened CIFAR-10 samples (top) and their corresponding reconstructions.

iments used a pre-trained decoder obtained via convolutional sparse coding with FISTA inference [25]. This decoder was used to perform the fixed-dictionary experiments. The decoder consists of 32-convolutional $3 \times 9 \times 9$ filters shown in Figure 4.2.

The top plot of Figure 4.5 presents the learning curves corresponding to the first five epochs of training LISTA architectures with 0,1,and 5 loops. The networks were trained to predict the sparse codes obtained using 10-iterations of FISTA inference by directly minimizing the distance in code space, namely:

$$\min_{W_e,S} ||Z_{FISTA} - LISTA_n(X; W_e, S)||$$

The codes obtained with FISTA corresponding to the samples in Figure 4.3 are visualized in Figure 4.4.

In the above Equation $LISTA_n$ refers to a LISTA network with n-loops of *shared* weights. The networks are trained via stochastic gradient descent with fixed learning rate. The top plot of Figure 4.5 presents the learning curves corresponding to the first five epochs of training LISTA architectures with 0,1,and 5 loops. The bottom plot tracks the loss corresponding to the codes output by the networks at the corresponding epochs.

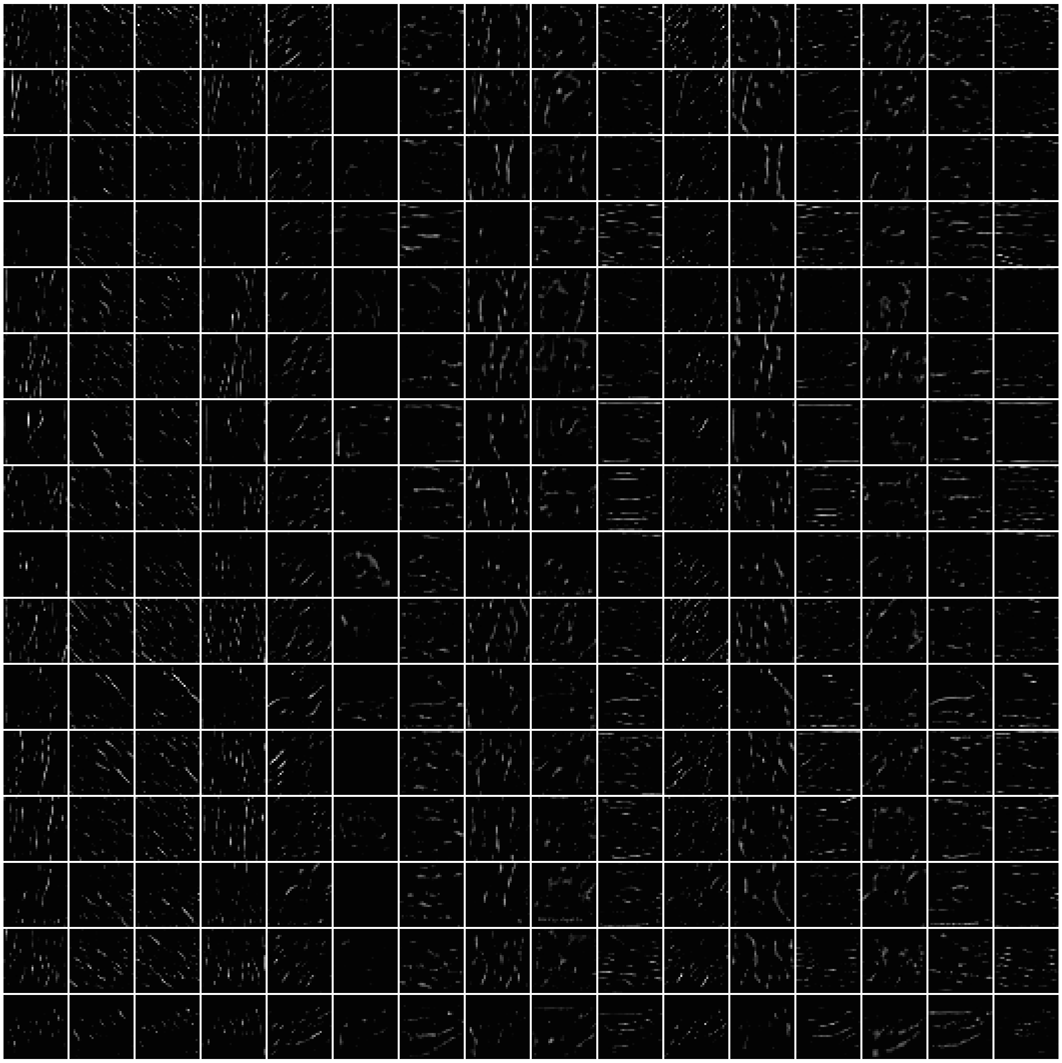


Figure 4.4: Sparse codes obtained with FISTA inference

The loss corresponding to 10-iterations of FISTA is shown as dashed red line. The loss is usually, but not always, commensurate with the distance in code space. Note that despite having the *same number of trainable parameters* LISTA networks with more loops seem to converge to a lower loss, or at least converge faster. However, the reason for this phenomena could be do to the fact that networks with more loops implicitly use a larger learning rate since gradients add in shared-weight networks. To eliminate this possibility, we use a coarse to fine random grid search to find the optimal learning rate for each network in the following experiments.

Assume that $Z = f_W(X)$, where $f_W()$ is a network with trainable weights W referred to as the encoder. The following experiments compare the performance of various architectures of $f_W()$ for minimizing Equation 4.1, with fixed and trainable decoder dictionaries. We compare the performance of the LISTA architecture to that of a more traditional deep ReLU networks, namely trainable convolutional layer interspersed with rectified linear point-wise non-linearities. Additionally we evaluate whether sharing weights is beneficial in networks trained for sparse inference. To this end, all multi-layer networks are trained with tied and untied weights. All experiments were repeated five times with different random initializations in order to measure the performance variance. In all experiments α, the L^1 weight in Equation 4.1, was set to 0.5.

4.2.1 Sparse Inference in a Fixed Dictionary

Figure 4.6 shows the lasso test loss corresponding the codes produced by the various architectures. The error bars correspond to the empirical standard deviation obtained by training each network from five different random intializations. The loss corresponding to the FISTA iterative inference algorithm (shown on the left side of the x-axis) can be considered as the lower bound on the test set. The next best codes are produced by

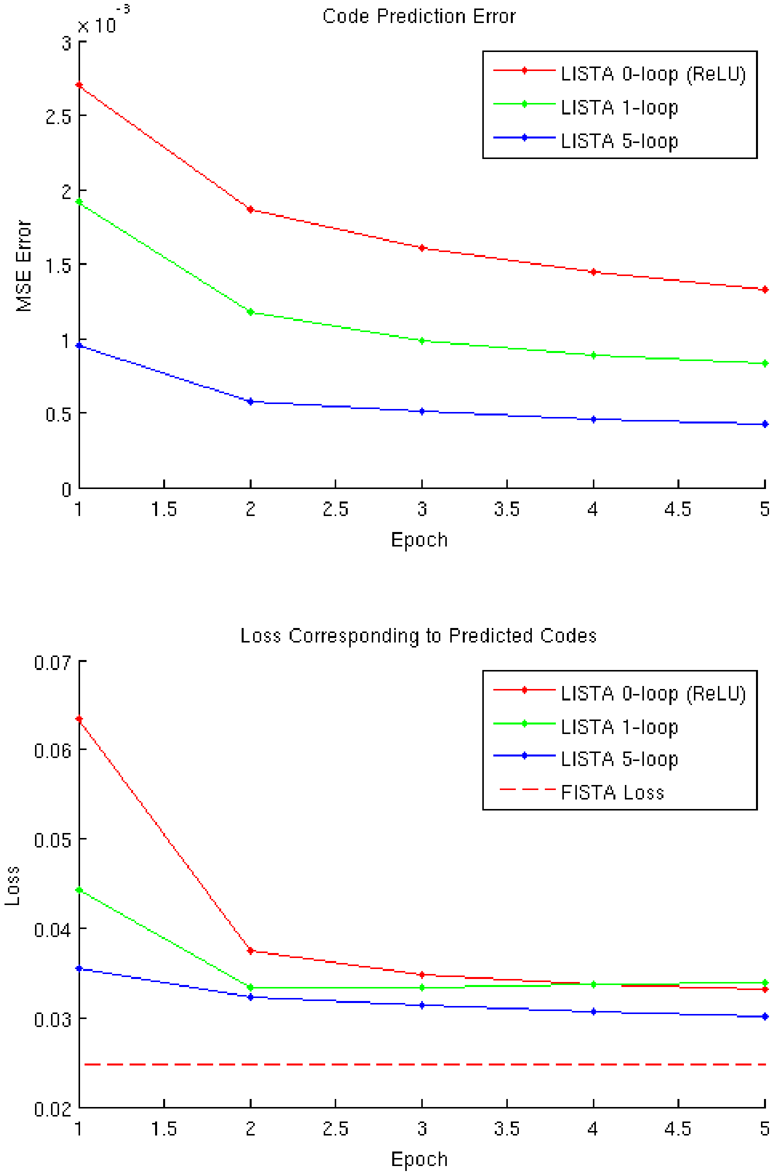


Figure 4.5: Sparse inference learning curves

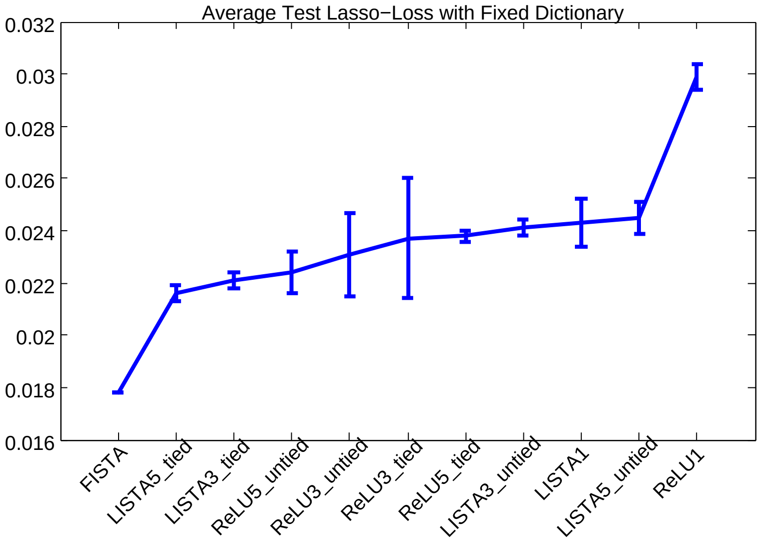


Figure 4.6: Lasso Loss with Fixed Dictionary

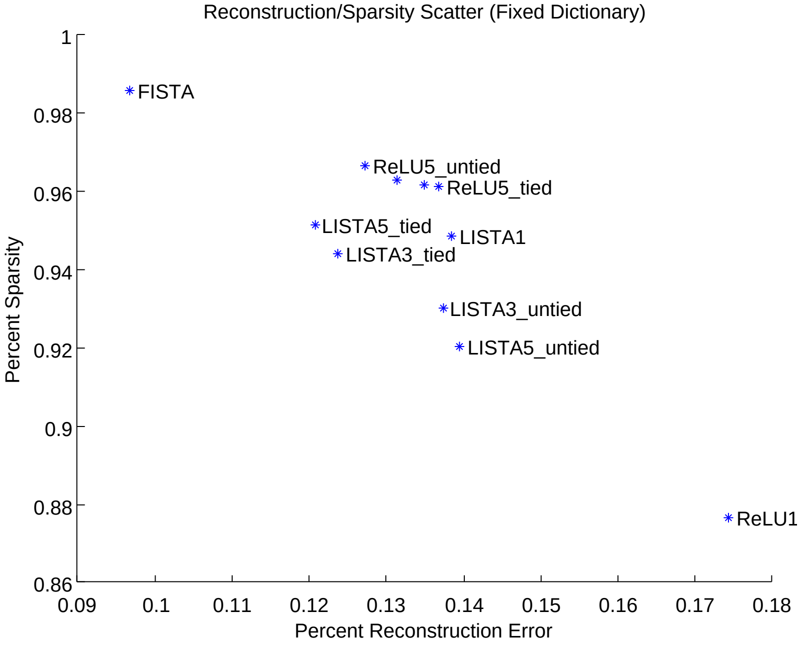


Figure 4.7: Reconstruction and Sparsity with Fixed Dictionary

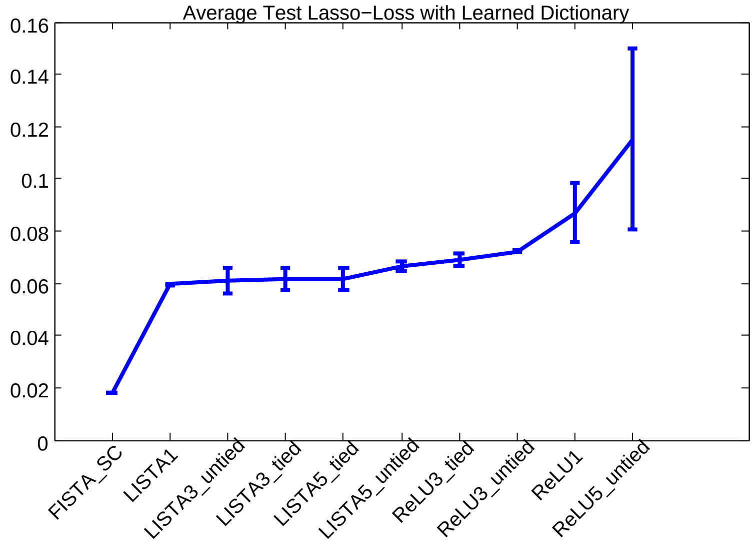


Figure 4.8: Lasso Loss with Learned Dictionary

the LISTA network with the most loops (five) and *shared* weights. ReLU networks had identical capacity as the LISTA networks, namely the W_e and S filter banks, with the skip connections of the LISTA networks removed. Figure 4.7 shows a scatter plot where each point once again corresponds to the codes produced by each network. The x-axis is the average percent reconstruction error, and the y-axis is the percent sparsity (proportion of activations that are zero). From these results, it can be concluded that indeed the LISTA architecture is superior to the ReLU network baseline. Moreover, shared weight LISTA networks outperform their untied weight counterparts despite having a smaller capacity.

4.2.2 Sparse Convolutional Auto-encoders

The following experiments involve minimizing Equation 4.1 with respect to the parameters of the encoder *and* W_d, jointly. This architecture is known as the "sparse autoencoder" [39]. In this setting both the parameters of the encoder *and* the decoder dictionary are initialized to random values. In order to avoid the scale ambiguity in which

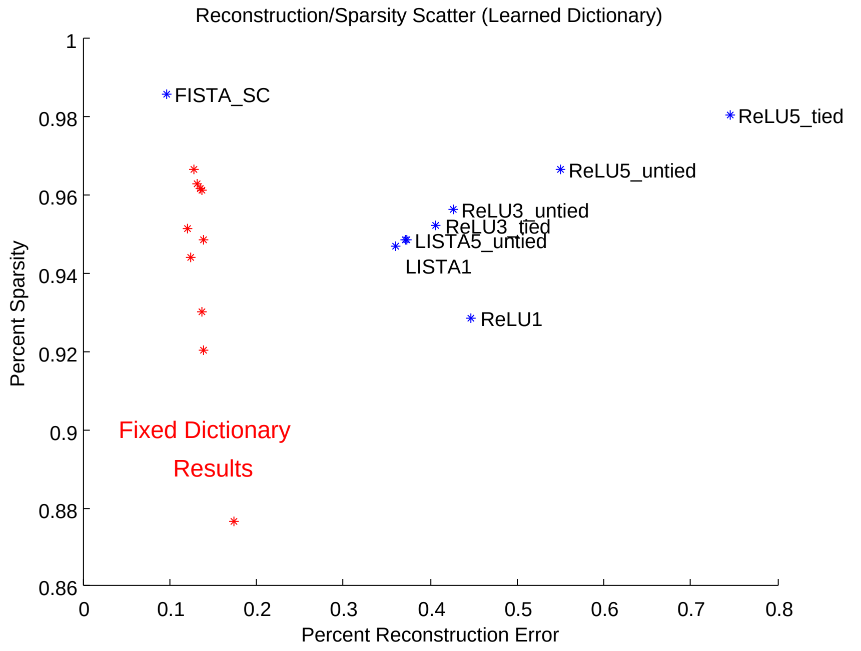


Figure 4.9: Reconstruction and Sparsity with Learned Dictionary

the encoder learns small weights to produce a small Z while the decoder learns large weights to recover the scale of the input, we use a normalized decoder where each kernel is renormalized to unit norm after every gradient update. Figure 4.8 shows the lasso loss on the test set corresponding to the dictionaries learned with the various encoder architectures. Note that the resulting losses are nearly three times larger in the learned dictionary case compared to those obtained with a pre-trained dictionary. This suggests that learning the encoder and decoder jointly converges to less optimal solutions. Indeed Figure 4.9 shows that the learned encoder/decoder pairs result in a larger reconstruction error than only training the encoder to perform inference in an already optimal dictionary. Furthermore, it is far less certain whether a multiple-loop LISTA architecture is the optimal encoder architecture in this setting.

Chapter 5

Learning Spatiotemporally Coherent Metrics

5.1 Introduction

Is it possible to characterize "good" representations without specifying a task a priori? If so, does there exist a set of generic priors which lead to these representations? In recent years state-of-the-art results from supervised learning suggest that the most powerful representations for solving specific tasks can be learned from the data itself. It has been hypothesized that large collections of unprocessed and unlabeled data can be used to learn generically useful representations. However the principles which would lead to these representations in the realm of unsupervised learning remain elusive. Temporal coherence is a form of weak supervision, which we exploit to learn generic signal representations that are stable with respect to the variability in natural video, including local deformations.

Our main assumption is that data samples that are temporal neighbors are also likely to be neighbors in the latent space. For example, adjacent frames in a video sequence are more likely to be semantically similar than non-adjacent frames. This assumption naturally leads to the slowness prior on features which was introduced in SFA ([49]).

This prior has been successfully applied to metric learning, as a regularizer in supervised learning, and in unsupervised learning ([19, 32, 49]). A popular assumption in unsupervised learning is that high dimensional data lies on a low dimensional manifold parametrized by the latent variables as in [4, 42, 46, 17]. In this case, temporal sequences can be thought of as one-dimensional trajectories on this manifold. Thus, an ensemble of sequences that pass through a common data sample have the potential to reveal the local latent variable structure within a neighborhood of that sample.

Non-linear operators consisting of a redundant linear transformation followed by a point-wise nonlinearity and a local pooling, are fundamental building blocks in deep convolutional networks. This is due to their capacity to generate local invariance while preserving discriminative information ([29, 6]). We justify that pooling operators are a natural choice for our unsupervised learning architecture since they induce invariance to local deformations. The resulting pooling auto-encoder model captures the main source of variability in natural video sequences, which can be further exploited by enforcing a convolutional structure. Experiments on YouTube data show that one can learn pooling representations with good discrimination and stability to observed temporal variability. We show that these features represent a metric which we evaluate on retrieval and classification tasks.

5.2 Contributions and Prior Work

The problem of learning temporally stable representations has been extensively studied in the literature, most prominently in Slow Feature Analysis (SFA) and Slow Subspace Analysis (SSA) ([49, 26, 23]). Works that learn slow features distinguish themselves mainly in three ways: (1) how the features are parametrized, (2) how the trivial (constant) solution is avoided, and (3) whether or not additional priors such as independence or sparsity are imposed on the learned features.

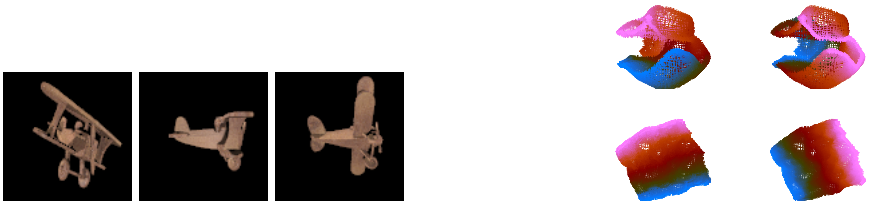
The features presented in SFA take the form of a nonlinear transformation of the input, specifically a quadratic expansion followed by a linear combination using learned weights optimized for slowness ([49]). This parametrization is equivalent to projecting onto a learned basis followed by L_2 pooling. The recent work by [30] uses features which are composed of projection onto a learned unitary basis followed by a local L_2 pooling in groups of two.

Slow feature learning methods also differ in the way that they avoid the trivial solution of learning to extract constant features. Constant features are perfectly slow (invariant), however they are not informative (discriminative) with respect to the input. All slow feature learning methods must make a trade-off between the discriminability and stability of the learned features in order to avoid trivial solutions. Slow Feature Analysis introduces two additional constraints, namely that the learned features must have unit variance and must be decorrelated from one another. In the work by [30], the linear part of the transformation into feature space is constrained to be unitary. Enforcing that the transform be unitary implies that it is invertible *for all inputs*, and not just the data samples. This unnecessarily limits the invariance properties of the transform and precludes the possibility of learning over-complete bases. Since the pooling operation following this linear transform has no trainable parameters, including this constraint is sufficient to avoid the trivial solution. Metric learning approaches ([19]) can be used to perform dimensionality reduction by optimizing a criteria which minimizes the distance between temporally adjacent samples in the transformed space, while repelling non-adjacent samples with a hinge loss, as explained in Section 5.3. The margin based contrastive term in DrLIM is explicitly designed to only avoid the constant solution and provides no guarantee on how informative the learned features are. Furthermore since distances grow exponentially due to the curse of dimensionality, metric based contrastive terms can be trivially satisfied in high dimensions.

Our approach uses a reconstruction criterion as a contrastive term. This approach is most similar to the one taken by [24] when optimizing group sparsity. In this work group-sparsity is replaced by slowness, and multiple layers of convolutional slow features are trained.

Several other studies combine the slowness prior with independence inducing priors [30, 8, 53]. For a detailed discussion on the connection between independence and sparsity see [22]. However, our model maximizes the sparsity of the representation *before* the pooling operator. Our model can be interpreted as a sparse auto-encoder additionally regularized by slowness through a local pooling operator.

In this work we introduce the use of convolutional pooling architectures for slow feature learning. At small spatial scales, local translations comprise the dominant source of variability in natural video; this is why many previous works on slowness learn mainly locally translation-invariant features ([49, 26, 30]). However, convolutional pooling architectures are locally translation-invariant by design, which allows our model to learn features that capture a richer class of invariances, beyond translation. Finally, we demonstrate that nontrivial convolutional dictionaries can be learned in the unsu-



(a) (b)

Figure 5.1: (a) Three samples from our rotating plane toy dataset. (b) Scatter plot of the dataset plotted in the output space of G_W at the start (top) and end (bottom) of training. The left side of the figure is colored by the yaw angle, and the right side by roll, 0° blue, 90° in pink.

pervised setting using only stochastic gradient descent (on mini-batches), despite their huge redundancy — that is, without resorting to alternating descent methods or iterative sparse inference algorithms.

5.3 Slowness as Metric Learning

coherence can be exploited by assuming a prior on the features extracted from the temporal data sequence. One such prior is that the features should vary slowly with respect to time. In the discrete time setting this prior corresponds to minimizing an L^p norm of the difference of feature vectors for temporally adjacent inputs. Consider a video sequence with T frames, if z_t represents the feature vector extracted from the frame at time t then the slowness prior corresponds to minimizing $\sum_{t=1}^{T} ||z_t - z_{t-1}||_p$. To avoid the degenerate solution $z_t = z_0$ for $t = 1...T$, a second term is introduced which encourages data samples that are *not* temporal neighbors to be separated by at least a distance of m-units in feature space, where m is known as the margin. In the temporal setting this corresponds to minimizing $max(0, m - ||z_t - z_{t'}||_p)$, where $|t - t'| > 1$. Together the two terms form the loss function introduced in [19] as a dimension reduction and data visualization algorithm known as DrLIM. Assume that there is a differentiable mapping from input space to feature space which operates on *individual* temporal samples. Denote this mapping by G and assume it is parametrized by a set of trainable coefficients denoted by W. That is, $z_t = G_W(x_t)$. The per-sample loss function can be written as:

$$L(x_t, x_{t'}, W) = \begin{cases} ||G_W(x_t) - G_W(x_{t'})||_p, & \text{if } |t - t'| = 1 \\ \max(0, m - ||G_W(x_t) - G_W(x_{t'})||_p) & \text{if } |t - t'| > 1 \end{cases} \tag{5.1}$$

In practice the above loss is minimized by constructing a ”Siamese” network ([5]) with shared weights whose inputs are pairs of samples along with their temporal indices. The loss is minimized with respect to the trainable parameters with stochastic gradient descent via back-propagation. To demonstrate the effect of minimizing Equation 5.1 on temporally coherent data, consider a toy data-set consisting of only one object. The data-set is generated by rotating a 3D model of a toy plane (Figure 5.1a) by 90° in one-degree increments around two-axes of rotation, generating a total of 8100 data samples. Input images (96×96) are projected into two-dimensional output space by the mapping G_W. In this example the mapping $G_W(X) : \mathbb{R}^{9216} \rightarrow \mathbb{R}^2$. We chose G_W to be a fully connected two layer neural network. In effect this data-set lies on an intrinsically two-dimensional manifold parametrized by two rotation angles. Since the sequence was generated by continuously rotating the object, temporal neighbors correspond to images of the object in similar configurations. Figure 5.1b shows the data-set plotted in the output space of G_W at the start (top row) and end (bottom row) of training. The left and right hand sides of Figure 5.1b are colorized by the two rotational angles, which are never explicitly presented to the network. This result implies that G_W has learned a mapping in which the latent variables (rotation angles) are linearized. Furthermore, the gradients corresponding to the two rotation angles are nearly orthogonal in the output space, which implies that the two features extracted by G_W are independent.

5.4 Slow Feature Pooling Auto-Encoders

The second contrastive term in Equation 5.1 only acts to avoid the degenerate solution in which G_W is a constant mapping, it does not guarantee that the resulting feature space is informative with respect to the input. This discriminative criteria only depends on pairwise distances in the representation space which is a geometrically weak notion in high dimensions. We propose to replace this contrastive term with a term that penalizes the reconstruction error of both data samples. Introducing a reconstruction terms not only prevents the constant solution but also acts to explicitly preserve information about the input. This is a useful property of features which are obtained using unsupervised learning; since the task to which these features will be applied is not known a priori, we would like to preserve as much information about the input as possible.

What is the optimal architecture of G_W for extracting slow features? Slow features are invariant to temporal changes by definition. In natural video and on small spatial scales these changes mainly correspond to local translations and deformations. Invariances to such changes can be achieved using appropriate pooling operators [6, 29]. Such operators are at the heart of deep convolutional networks (ConvNets), currently the most successful supervised feature learning architectures [28]. Inspired by these observations, let G_{W_e} be a two stage encoder comprised of a learned, generally over-complete, linear map (W_e) and rectifying nonlinearity $f(\cdot)$, followed by a local pooling. Let the N hidden activations, $h = f(W_e x)$, be subdivided into K potentially overlapping neighborhoods denoted by P_i. Note that biases are absorbed by expressing the input x in homogeneous coordinates. Feature z_i produced by the encoder for the input at time t can be expressed as $G^i_{W_e}(t) = \|h_t\|_p^{P_i} = \left(\sum_{j\in P_i} h_{tj}^p\right)^{\frac{1}{p}}$. Training through a local pooling operator enforces a local topology on the hidden activations, inducing units that are pooled together to learn complimentary features. In the following experiments we will use $p = 2$. Although it has recently been shown that it is possible to recover the input when W_e is sufficiently redundant, reconstructing from these coefficients corresponds to solving a phase recovery problem [7] which is not possible with a simple inverse mapping, such as a linear map W_d. Instead of reconstructing from z we reconstruct from the hidden representation h. This is the same approach taken when training group-sparse auto-encoders [24]. In order to promote sparse activations in the case of over-complete bases we additionally add a sparsifying L_1 penalty on the hidden activations. Including the rectifying nonlinearity becomes critical for learning sparse inference in a hugely redundant dictionary, e.g. convolutional dictionaries [18]. The complete loss functional is:

$$L(x_t, x_{t'}, W) = \sum_{\tau=\{t,t'\}} \left(\|W_d h_\tau - x_\tau\|^2 + \alpha|h_\tau|\right) + \beta \sum_{i=1}^{K} \left|\|h_t\|^{P_i} - \|h_{t'}\|^{P_i}\right| \tag{5.2}$$

Figure 5.3 shows a convolutional version of the proposed architecture and loss. This combination of loss and architecture can be interpreted as follows: the sparsity penalty induces the first stage of the encoder, $h = f(W_e x)$, to approximately infer sparse codes in the analysis dictionary W_e; the slowness penalty induces the formation of pool groups whose output is stable with respect to temporal deformations. In other words, the first stage partitions the input space into disjoint linear subspaces and the second stage recombines these partitions into temporally stable groups. This can be seen as a sparse auto-encoder whose pooled codes are additionally regularized by slowness.

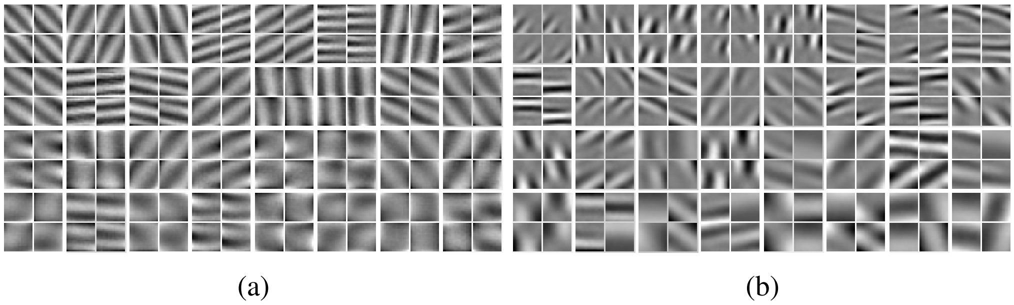


Figure 5.2: Pooled decoder dictionaries learned without (a) and with (b) the L_1 penalty using (6.1).

5.4.1 Fully-Connected Architecture

To gain an intuition for the properties of the minima of Equation 6.1 for natural data, an auto-encoder was trained on a small dataset consisting of natural movie patches. This data set consists of approximately 170,000, 20×20 gray scale patches extracted from full resolution movies. Minimizing Equation 6.1 with $\alpha = 0$ results in the learned decoder basis shown in Figure 5.2a. Here a dictionary of 512 basis elements was trained whose outputs were pooled in non-overlapping groups of four resulting in 128 output features. Only the slowest 32 groups are shown in Figure 5.2a. The learned dictionary has a strong resemblance to the two-dimensional Fourier basis, where most groups are comprised of phase shifted versions of the same spatial frequency. Since translations are an invariant of the local modulus of the Fourier transform, the result of this experiment is indicative of the fact that translations are the principal source of variation at small spatial scales. Minimizing Equation 6.1 with $\alpha > 0$ results in a more localized basis depicted in Figure 5.2b. This basis is more consistent with a local deformation model as opposed to a global one.

5.4.2 Convolutional Architecture

By replacing all linear operators in our model with convolutional filter banks and including spatial pooling, translation invariance need not be learned [29]. In all other respects the convolutional model is conceptually identical to the fully connected model described in the previous section. One important difference between fully-connected and convolutional dictionaries is that the later can be massively over-complete, making sparse inference potentially more challenging. Nevertheless we found that non-trivial dictionaries (see Figure 5.5d) can be learned using purely stochastic optimization, that is, without a separate sparse inference phase. Let the linear stage of the encoder consist of a filter bank which takes C input feature maps (corresponding to the 3 color channels for the first stage) and produces D output feature maps. Correspondingly, the convolutional decoder transforms these D feature maps back to C color channels. In the convolutional setting slowness is measured by subtracting corresponding spatial locations in temporally adjacent feature maps. In order to produce slow features a convolutional network must compensate for the motion in the video sequence by producing *spatially* aligned activations for *temporally* adjacent samples. In other words, in order to produce slow features the network must implicitly learn to track common patterns by learning features which are invariant to the deformations exhibited by these patterns in the temporal sequence. The primary mechanism for producing these invariances is pooling in space and across features [14]. Spatial pooling induces local translation invariance. Pooling across feature maps allows the network to potentially learn feature groups that are stable with respect to more general deformations. Intuitively, maximizing slowness in a convolutional architecture leads to *spatiotemporally* coherent features.

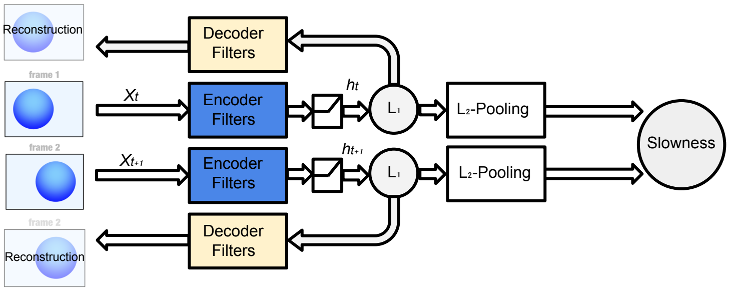


Figure 5.3: Block diagram of the Siamese convolutional model trained on pairs of frames.

5.5 Experimental Results

To verify the connection between slowness and metric learning, we evaluate the metric properties of the learned features. It is well known that distance in the extrinsic (input pixel) space is not a reliable measure of semantic similarity. Maximizing slowness corresponds to minimizing the distance between adjacent frames in code space, therefore neighbors in code space should correspond to temporal neighbors. This claim can be tested by computing the nearest neighbors to a query frame in code space, and verifying whether they correspond to its temporal neighbors. However, the features must also be discriminative so as not to collapse temporally distant samples. In order to make this trade-off in a principled manner, a dataset comprised of short natural scenes was collected. Hyper-parameters are selected which maximize the so called ”temporal coherence” of the features which define the metric. We define the temporal coherence of a metric $G_W(\cdot)$ as the area under the precision-recall curve, where precision is defined as the proportion of the nearest neighbors that come from the same scene, and recall is defined as the proportion of frames recalled from that scene. In our experiments, we used the middle frame from each scene as the query.

However, temporal coherence can be a very weak measure of discriminability; it merely requires that scenes be easy to disambiguate in feature space. If the scenes are quite distinct, then maximizing temporal coherence directly can lead to weakly discriminative features (e.g. color histograms can exhibit good temporal coherence). We therefore evaluate the learned features on a more demanding task by assessing how well the metric learned from the YouTube dataset transfers to a classification task on the CIFAR-10 dataset. Average class-based precision is measured in feature space by using the test set as the query images and finding nearest neighbors in the training set. Precision is defined as the proportion of nearest neighbors that have the same label. As on the YouTube dataset we evaluate the average precision for the nearest 40 neighbors. The CIFAR dataset contains considerably more interclass variability than the scenes in our YouTube dataset, nevertheless many class instances are visually similar.

approximately $150,000$ frames extracted from YouTube videos. Of these, approximately $20,000$ frames were held out for testing. The training and test set frames were collected from separate videos. The videos were automatically segmented into scenes of variable length (2-40 frames) by detecting large L_2 changes between adjacent frames. Each color frame was down-sampled to a 32×32 spatial resolution and the entire dataset was ZCA whitened [27]. Six scenes from the test set are shown in Figure 5.4 where the first scene (top row) is incorrectly segmented.

We compare the features learned by minimizing the loss in Equation 6.1 with the features learned by minimizing DrLIM (Equation 5.1) and group sparsity (Equation 5.3) losses. Once trained, the convolution, rectification, and pooling stages are used to transform the dataset into the feature space. We use cosine distance in feature space to determine the nearest neighbors and select hyperparmeters for each method which maximize the temporal coherence measure.

We trained two layers of our model using greedy layer-wise training [4]. The first layer model contains a filter bank consisting of 64 kernels with 9×9 spatial support. The first L_2 pooling layer computes the local modulus volumetrically, that is *across* feature maps in non-overlapping groups of four and spatially in 2×2 non-overlapping neighborhoods. Thus the output feature vector of the first stage (z_1) has dimensions $16 \times 16 \times 16$ (4096). Our second stage consists of 64 5×5 convolutional filters, and performs 4×4 spatial pooling producing a second layer code (z_2) of dimension $64 \times 4 \times 4$ (1024). The output of the second stage corresponds to a dimension reduction by a factor of three relative to the input dimension.

Identical one and two-layer architectures were trained using the group sparsity prior, similar to [24]. As in the slowness model, the two layer architecture was trained greedily. Using the same notation as Equation 6.1, the corresponding loss can be written as:

$$L(x_t, W) = \sum_{\tau} ||W_d h_\tau - x_\tau||^2 + \alpha ||h_\tau||^{P_i} \tag{5.3}$$

Finally, identical one and two-layer architectures were also trained by minimizing the DrLIM loss in Equation 5.1. Negative pairs, corresponding to temporally non-adjacent frames, were independently selected at random. In order to achieve the best temporal precision-recall performance, we found that each mini-batch should consist of a large proportion of negative to positive samples (at least five-to-one). Unlike the auto-encoder methods, the two layer architecture was trained jointly rather than greedily.

results on the YouTube dataset for a single frame (left column) in eight spaces. The top row shows the nearest neighbors in pixel space. The second row shows the nearest neighbors in pixel space after ZCA whitening. The next six rows show the nearest neighbors in feature space for one and two layer feature transformations learned with slowness, group sparsity, and DrLIM. The resulting first-layer filters and precision-recall

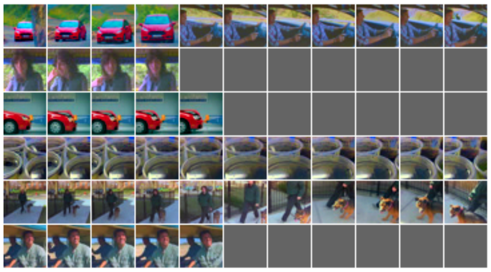


Figure 5.4: Six scenes from our YouTube dataset

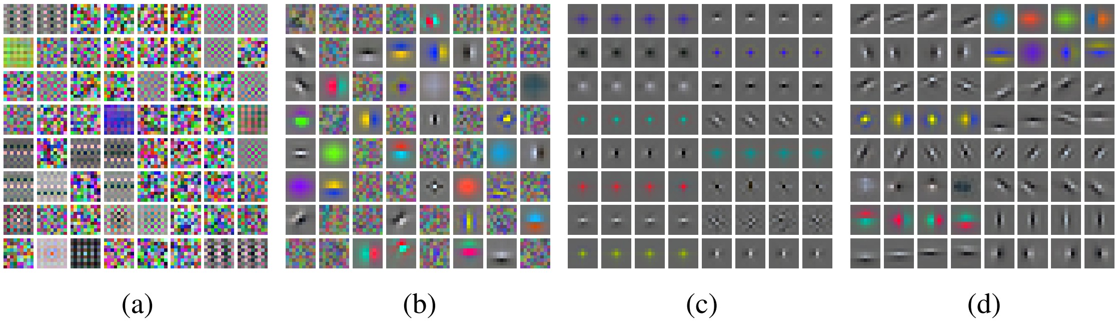
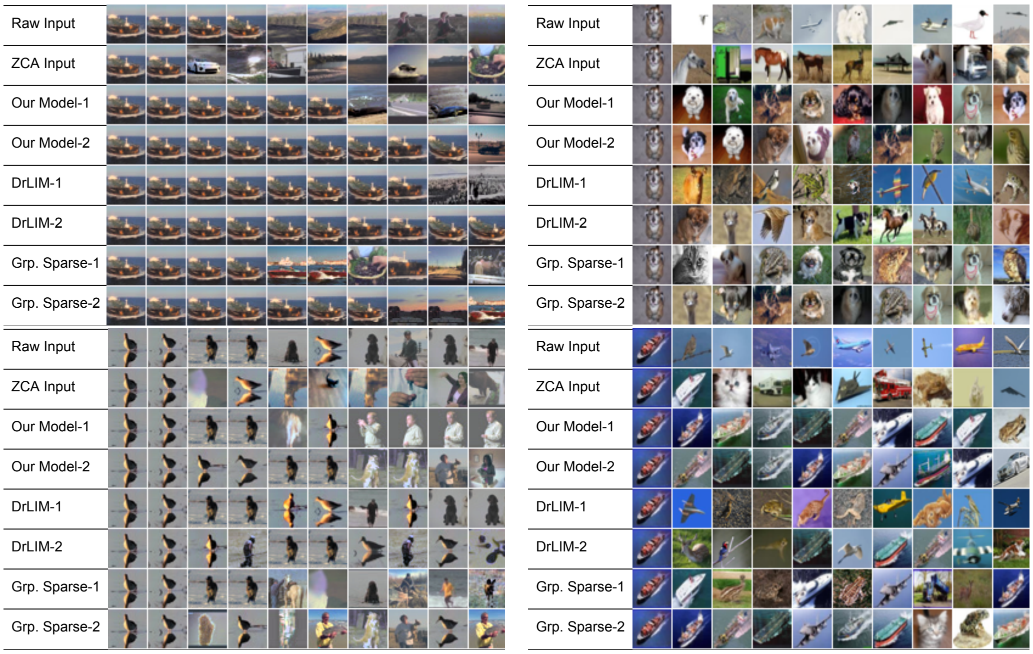


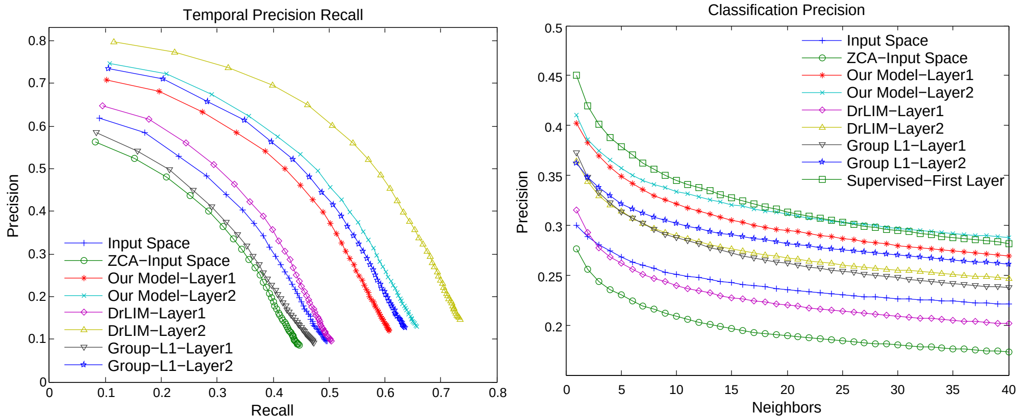
Figure 5.5: Pooled convolutional dictionaries (decoders) learned with: (a) DrLIM and (b) sparsity only, (c) group sparsity, and (d) sparsity and slowness. Groups of four features that were pooled together are depicted as horizontally adjacent filters.

curves are shown in Figures 6.3 and 5.7, respectively. Figures 5.5b and 5.5d show the decoders of two one-layer models trained with $\beta = 0, 2$, respectively, and a constant value of α. The filter bank trained with $\beta = 0$ exhibits no coherence within each pool group; the filters are not visually similar nor do they tend to co-activate at spatially neighboring locations. Most groups in the filter bank trained with slowness tend be visually similar, corresponding to similar colors and/or geometric structures. The features learned by minimizing the DrLIM loss (Equation 5.1), which more directly optimizes temporal coherence, have much more high frequency content than the filters learned with any of the auto-encoder methods. Nevertheless, some filters within the same pool group exhibit similar geometric and color structure (Figure 5.5a). The features learned with a group-sparsity regularizer leads to nearly identical features (and nearly identical



(a) (b)

Figure 5.6: Query results in the (a) video and (b) CIFAR-10 datasets. Each row corresponds to a different feature space in which the queries were performed; numbers (1 or 2) denote the number of convolution-pooling layers.



(a) (b)

Figure 5.7: Precision-Recall curves corresponding to the YouTube (a) and CIFAR-10 (b) dataset.

Model	Optimization	Temporal AUC	Class AUC
Our Model Layer1	—	0.262	0.296
Our Model Layer2	Greedy	0.300	**0.310**
DrLIM Layer1	—	0.188	0.221
DrLIM Layer2	Joint	**0.378**	0.268
Group L_1 Layer1	—	0.231	0.266
Group L_1 Layer2	Greedy	0.285	0.281

activations) within each pool group (Figure 5.5c). This is not surprising because group sparsity promotes co-activation of the features within each pool group, by definition. We have also tried including an individual sparsity prior, as in Equation 6.1, in order to encourage independence among the pooled features. However this has lead to significantly worse temporal-coherence performance.

Figure 5.6b shows the result of two queries in the CIFAR-10 dataset. The corresponding precision-recall curves are shown in Figure 5.7b. One-layer DrLIM (4096 dimensional) exhibit poor performance in both the temporal and class-based recall tasks. In contrast, jointly trained two-layer DrLIM features (1024 dimensional) exhibit excellent temporal coherence, outperforming all other models by a large margin. Although better than the first layer, second layer features perform significantly worse on the CIFAR task than even the first-layer features learned by our model. Furthermore, the nearest neighbors in both the one and two-layer feature spaces learned with DrLIM are often neither visually nor semantically similar (see Figure 5.6b). The conclusion which can be drawn from this result is that ***directly maximizing temporal coherence alone is not a sufficient condition for achieving a semantically (or even visually) coherent features***. However, combining it with reconstruction and sparsity, as in our model, yields the most semantically discriminative features. Although significantly better than the features learned with DrLIM, the features learned with group sparsity exhibit slightly weaker temporal coherence and significantly worse class-based recall. Note that since all the features within a pool group are practically identical, the invariants captured by the pool groups are limited to local translations due to the spatial pooling. As a final comparison, we trained a four layer ConvNet with supervision on CIFAR-10, this network achieved approximately 80% classification accuracy on the test set. The architecture of the first two stages of the ConvNet is identical to the architecture of the first and second unsupervised stages. The precision curve corresponding to the first layer of the ConvNet is shown in Figure 5.7b, which is matched by our-model's second layer at high recall.

5.6 Conclusion

Video data provides a virtually infinite source of information to learn meaningful and complex visual invariances. While temporal slowness is an attractive prior for good visual features, in practice it involves optimizing conflicting objectives that balance invariance and discriminability. In other words, perfectly slow features cannot be informative. An alternative is to replace the small temporal velocity prior with small temporal acceleration, leading to a criteria that *linearizes* observed variability. The resulting representation offers potential advantages, such as extraction of both locally invariant and locally covariant features. Although pooling representations are widespread in visual and audio recognition architectures, much is left to be understood. In particular, a major question is how to learn a stacked pooling representation, such that its invariance properties are boosted while controlling the amount of information lost at each layer. This could be possible by replacing the linear decoder of the proposed model with a nonlinear decoder which can be used to reconstruct the input from pooled representations. Slow feature learning is merely one way to learn from temporally coherent data. In this work we have provided an auto-encoder formulation of the problem and shown that the resulting features are more stable to naturally occurring temporal variability, while maintaining discriminative power.

Chapter 6

Learning to Linearize under Uncertainty

6.1 Introduction

The recent success of deep feature learning in the supervised setting has inspired renewed interest in feature learning in weakly supervised and unsupervised settings. Recent findings in computer vision problems have shown that the representations learned for one task can be readily transferred to others [36], which naturally leads to the question: does there exist a generically useful feature representation, and if so what principles can be exploited to learn it?

Recently there has been a flurry of work on learning features from video using varying degrees of supervision [48][41][47]. Temporal coherence in video can be considered as a form of weak supervision that can be exploited for feature learning. More precisely, if we assume that data occupies some low dimensional "manifold" in a high dimensional space, then videos can be considered as one-dimensional trajectories on this manifold parametrized by time. Many unsupervised learning algorithms can be viewed as various parameterizations (implicit or explicit) of the data manifold [4]. For instance, sparse coding implicitly assumes a locally linear model of the data manifold [34]. In this work, we assume that deep convolutional networks are good parametric models for natural data. Parameterizations of the data manifold can be learned by training these networks to *linearize* short temporal trajectories, thereby implicitly learning a local parametrization.

In this work we cast the linearization objective as a frame *prediction* problem. As in many other unsupervised learning schemes, this necessitates a generative model. Several recent works have also trained deep networks for the task of frame prediction [41][48][47]. However, unlike other works that focus on prediction as a final objective, in this work prediction is regarded as a proxy for learning representations. We introduce a loss and architecture that addresses two main problems in frame prediction: (1) minimizing L^2 error between the predicted and actual frame leads to unrealistically blurry predictions, which potentially compromises the learned representation, and (2) copying the most recent frame to the input seems to be a hard-to-escape trap of the objective function, which results in the network learning little more than the identity function. We argue that the source of blur partially stems from the inherent unpredictability of natural data; in cases where multiple valid predictions are plausible, a deterministic network will learn to average between all the plausible predictions. To address the first problem we introduce a set of latent variables that are non-deterministic functions of the input, which are used to explain the unpredictable aspects of natural videos. The second problem is addressed by introducing an architecture that explicitly formulates the prediction in the linearized feature space.

The paper is organized as follows. Section 6.2 reviews relevant prior work. Section

6.3 introduces the basic architecture used for learning linearized representations. Subsection 6.3.1 introduces “phase-pooling”–an operator that facilitates linearization by inducing a topology on the feature space. Subsection 6.3.2 introduces a latent variable formulation as a means of learning to linearize under uncertainty. Section 6.4 presents experimental results on relatively simple datasets to illustrate the main ideas of our work. Finally, Section **??** offers directions for future research.

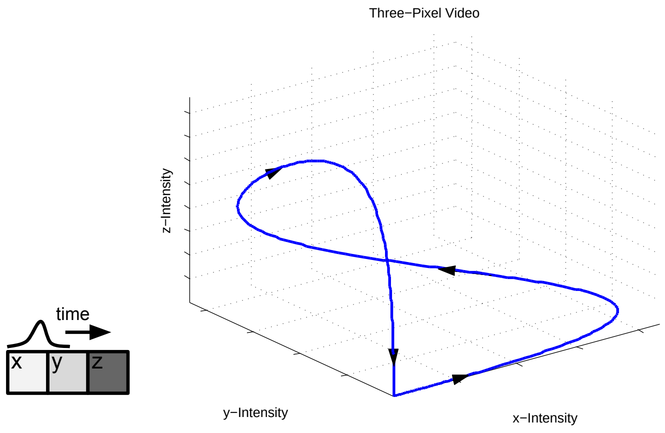
6.2 Prior Work

This work was heavily inspired by the philosophy revived by Hinton et al. [20], which introduced “capsule” units. In that work, an equivariant representation is learned by the capsules when the true latent states were provided to the network as implicit targets. Our work allows us to move to a more unsupervised setting in which the true latent states are not only unknown, but represent completely arbitrary qualities. This was made possible with two assumptions: (1) that temporally adjacent samples also correspond to neighbors in the latent space, (2) predictions of future samples can be formulated as *linear* operations in the latent space. In theory, the representation learned by our method is very similar to the representation learned by the “capsules”; this representation has a locally stable *“what”* component and a locally linear, or equivariant *“where”* component. Theoretical properties of linearizing features were studied in [9].

Several recent works propose schemes for learning representations from video which use varying degrees of supervision[41][48][47][15]. For instance, [47] assumes that the pre-trained network from [28] is already available and training consists of learning to mimic this network. Similarly, [48] learns a representation by receiving supervision from a tracker. This work is more closely related to fully unsupervised approaches for learning representations from video such as [15][26][8][49][32]. It is most related to [41] which also trains a decoder to explicitly predict video frames. Our proposed architecture was inspired by those presented in in [38] and [52].

6.3 Learning Linearized Representations

Our goal is to obtain a representation of each input sequence that varies linearly in time by transforming each frame *individually*. Furthermore, we assume that this transformation can be learned by a deep, feed forward network referred to as the *encoder*, denoted by the function F_W. Denote the code for frame x^t by $z^t = F_W(x^t)$. Assume that the dataset is parameterized by a temporal index t so it is described by the sequence $X = \{..., x^{t-1}, x^t, x^{t+1}, ...\}$ with a corresponding feature sequence produced by the encoder $Z = \{..., z^{t-1}, z^t, z^{t+1}, ...\}$. Thus our goal is to train F_W to produce a sequence Z whose average local curvature is smaller than sequence X. A scale invariant local measure of curvature is the cosine distance between the two vectors formed by three temporally adjacent samples. However, minimizing the curvature directly can result in the trivial solutions: $z_t = ct \ \forall \ t$ and $z_t = c \ \forall \ t$. These solutions are trivial because they are virtually *uninformative* with respect to the input x^t and therefore cannot be a *meaningful representation of the input*. To avoid this solution, we also minimize the prediction error in the input space. The predicted frame is generated in two steps: (i) linearly extrapolation in code space to obtain a predicted code $\hat{z}^{t+1} = \mathbf{a}[z^t \ \ z^{t-1}]^T$ followed by (ii) a *decoding* with G_W, which generates the predicted frame $\hat{x}^{t+1} = G_W(\hat{z}^{t+1})$. For example, if $\mathbf{a} = [2, -1]$ the predicted code $\hat{z}^{t+1}$ corresponds to a constant speed linear extrapolation of z^t and z^{t-1}. The L^2 prediction error is minimized by jointly training the encoder and decoder networks. Note that minimizing prediction error alone will not necessarily



(a) (b)

Figure 6.1: (a) A video generated by translating a Gaussian intensity bump over a three pixel array (x,y,z), (b) the corresponding manifold parametrized by time in three dimensional space

lead to low curvature trajectories in Z since the decoder is unconstrained; the decoder may learn a many to one mapping which maps different codes to the same output image without forcing them to be equal. To prevent this, we add an explicit curvature penalty to the loss, corresponding to the cosine distance between $(z^t - z^{t-1})$ and $(z^{t+1} - z^t)$. The complete loss to minimize is:

$$L = \frac{1}{2}||G_W(\mathbf{a}\begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T) - x^{t+1}||_2^2 - \lambda \frac{(z^t - z^{t-1})^T (z^{t+1} - z^t)}{||z^t - z^{t-1}||||z^{t+1} - z^t||} \tag{6.1}$$

This feature learning scheme can be implemented using an autoencoder-like network with shared encoder weights.

6.3.1 Phase Pooling

Thus far we have assumed a generic architecture for F_W and G_W. We now consider custom architectures and operators that are particularly suitable for the task of linearization. To motivate the definition of these operators, consider a video generated

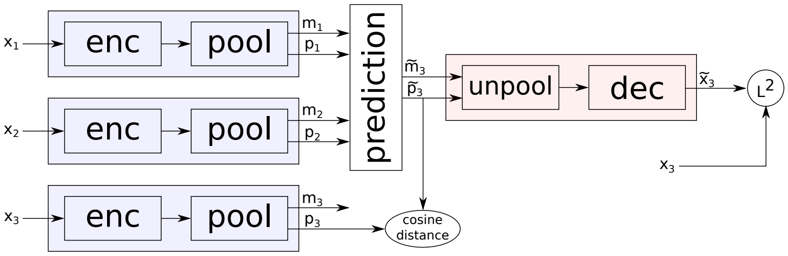


Figure 6.2: The basic linear prediction architecture with shared weight encoders

by translating a Gaussian “intensity bump” over a three pixel region at constant speed. The video corresponds to a one dimensional manifold in three dimensional space, i.e. a curve parameterized by time (see Figure 6.1). Next, assume that some convolutional feature detector fires only when centered on the bump. Applying the *max*-pooling operator to the activations of the detector in this three-pixel region signifies the presence of the feature somewhere in this region (*i.e. the “what”*). Applying the *argmax* operator over the region returns the position (*i.e. the “where”*) with respect to some local coordinate frame defined over the pooling region. This position variable varies linearly as the bump translates, and thus parameterizes the curve in Figure 6.1b. These two channels, namely the *what* and the *where*, can also be regarded as generalized magnitude m and phase $\mathbf{p}$, corresponding to a factorized representation: the magnitude represents the active set of parameters, while the phase represents the set of local coordinates in this active set. We refer to the operator that outputs both the *max* and *argmax* channels as the “phase-pooling” operator.

In this example, spatial pooling was used to linearize the translation of a fixed feature. More generally, the phase-pooling operator can locally linearize arbitrary transformations if pooling is performed not only spatially, but also across features in some topology.

In order to be able to back-propagate through $\mathbf{p}$, we define a soft version of the *max* and *argmax* operators within each pool group. For simplicity, assume that the encoder has a fully convolutional architecture which outputs a set of feature maps, possibly of a different resolution than the input. Although we can define an arbitrary topology in feature space, for now assume that we have the familiar three-dimensional spatial feature map representation where each activation is a function $z(f, x, y)$, where x and y correspond to the spatial location, and f is the feature map index. Assuming that the feature activations are positive, we define our soft *"max-pooling"* operator for the k^{th} neighborhood N_k as:

$$m_k = \sum_{N_k} z(f, x, y) \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f\prime,x\prime,y\prime)}} \approx \max_{N_k} z(f, x, y), \tag{6.2}$$

where $\beta \geq 0$. Note that the fraction in the sum is a softmax operation (parametrized by β), which is positive and sums to one in each pooling region. The larger the β, the closer it is to a unimodal distribution and therefore the better m_k approximates the *max* operation. On the other hand, if $\beta = 0$, Equation 6.2 reduces to *average-pooling*. Finally, note that m_k is simply the expected value of z (restricted to N_k) under the softmax distribution.

Assuming that the activation pattern within each neighborhood is approximately unimodal, we can define a soft versions of the *argmax* operator. The vector $\mathbf{p}_k$ approximates the local coordinates in the feature topology at which the max activation value occurred. Assuming that pooling is done volumetrically, that is, spatially *and* across features, $\mathbf{p}_k$ will have three components. In general, the number of components in $\mathbf{p}_k$ is equal to the dimension of the topology of our feature space induced by the pooling neighborhood. The dimensionality of $\mathbf{p}_k$ can also be interpreted as the *maximal intrinsic dimension of the data*. If we define a local standard coordinate system in each pooling volume to be bounded between -1 and +1, the soft "*argmax-pooling*" operator is defined by the vector-valued sum:

$$\mathbf{p}_k = \sum_{N_k} \begin{bmatrix} f \\ x \\ y \end{bmatrix} \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f\prime,x\prime,y\prime)}} \approx \underset{N_k}{\arg\max}\, z(f,x,y), \tag{6.3}$$

where the indices f, x, y take values from -1 to 1 in equal increments over the pooling region. Again, we observe that $\mathbf{p}_k$ is simply the expected value of $\begin{bmatrix} f & x & y \end{bmatrix}^T$ under the softmax distribution.

The phase-pooling operator acts on the output of the encoder, therefore it can simply be considered as the last encoding step. Correspondingly we define an "*un-pooling*" operation as the first step of the decoder, which produces reconstructed activation maps by placing the magnitudes m at appropriate locations given by the phases $\mathbf{p}$.

Because the phase-pooling operator produces both magnitude and phase signals for each of the two input frames, it remains to define the predicted magnitude and phase of the third frame. In general, this linear extrapolation operator can be learned, however "hard-coding" this operator allows us to place implicit priors on the magnitude and phase channels. The predicted magnitude and phase are defined as follows:

$$m^{t+1} = \tfrac{m^t + m^{t-1}}{2} \tag{6.4}$$

$$\mathbf{p}^{t+1} = 2\mathbf{p}^t - \mathbf{p}^{t-1} \tag{6.5}$$

Predicting the magnitude as the mean of the past imposes an implicit stability prior on m, i.e. the temporal sequence corresponding to the m channel should be stable between adjacent frames. The linear extrapolation of the phase variable imposes an implicit linear prior on **p**. Thus such an architecture produces a factorized representation composed of a locally stable m and locally linearly varying **p**. When phase-pooling is used curvature regularization is only applied to the **p** variables. The full prediction architecture is shown in Figure 6.2.

6.3.2 Addressing Uncertainty

Natural video can be inherently unpredictable; objects enter and leave the field of view, and out of plane rotations can also introduce previously invisible content. In this case, the prediction should correspond to the *most likely outcome* that can be learned by training on similar video. However, if multiple outcomes are present in the training set then minimizing the L^2 distance to these multiple outcomes induces the network to predict the *average outcome*. In practice, this phenomena results in blurry predictions and may lead the encoder to learn a less discriminative representation of the input. To address this inherent unpredictability we introduce latent variables δ to the prediction architecture that are not deterministic functions of the input. These variables can be adjusted *using the target* x^{t+1} in order to minimize the prediction L^2 error. The interpretation of these variables is that they explain all aspects of the prediction that are not captured by the encoder. For example, δ can be used to switch between multiple, equally likely predictions. It is important to control the capacity of δ to prevent it from explaining the entire prediction on its own. Therefore δ is restricted to act only as a correction term in the code space output by the encoder. To further restrict the capacity of δ we enforce that $dim(\delta) \ll dim(z)$. More specifically, the δ-corrected code is defined as:

$$\hat{z}_{\delta}^{t+1} = z^t + (W_1\delta) \odot \mathbf{a}\begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T \tag{6.6}$$

Where W_1 is a trainable matrix of size $dim(\delta) \times dim(z)$, and $\odot$ denotes the component-wise product. During training, δ is inferred (using gradient descent) for each training sample by minimizing the loss in Equation 6.7. The corresponding adjusted $\hat{z}_{\delta}^{t+1}$ is then used for back-propagation through W and W_1. At test time δ can be selected via sampling, assuming its distribution on the training set has been previously estimated.

$$L = \min_{\delta} \|G_W(\hat{z}_{\delta}^{t+1}) - x^{t+1}\|_2^2 - \lambda \frac{(z^t - z^{t-1})^T (z^{t+1} - z^t)}{\|z^t - z^{t-1}\| \|z^{t+1} - z^t\|} \tag{6.7}$$

The following algorithm details how the above loss is minimized using stochastic gradient descent:

Algorithm 1 Minibatch stochastic gradient descent training for prediction with uncertainty. The number of δ-gradient descent steps (k) is treated as a hyper-parameter.

```
for number of training epochs do
    Sample a mini-batch of temporal triplets $\{x^{t-1}, x^t, x^{t+1}\}$
    Set $\delta_0 = 0$
    Forward propagate $x^{t-1}, x^t$ through the network and obtain the codes $z^{t-1}, z^t$ and the
    prediction $\hat{x}_0^{t+1}$
    for $i$ =1 to $k$ do
        Compute the $L^2$ prediction error
        Back propagate the error through the decoder to compute the gradient $\frac{\partial L}{\partial \delta^{i-1}}$
        Update $\delta_i = \delta_{i-1} - \alpha \frac{\partial L}{\partial \delta^{i-1}}$
        Compute $\hat{z}_{\delta_i}^{t+1} = z^t + (W_1 \delta_i) \odot \mathbf{a} \begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T$
        Compute $\hat{x}_i^{t+1} = G_W(z_{\delta^i}^{t+1})$
    end for
    Back propagate the full encoder/predictor loss from Equation 6.7 using $\delta_k$, and
    update the weight matrices $W$ and $W_1$
end for
```

When phase pooling is used we allow δ to only affect the phase variables in Equation 6.5, this further encourages the magnitude to be stable and places all the uncertainty in the phase.

6.4 Experiments

The following experiments evaluate the proposed feature learning architecture and loss. In the first set of experiments we train a shallow architecture on natural data and visualize the learned features in order gain a basic intuition. In the second set of experiments we train a deep architecture on simulated movies generated from the NORB dataset. By generating frames from interpolated and extrapolated points in code space we show that a linearized representation of the input is learned. Finally, we explore the role of uncertainty by training on only partially predictable sequences, we show that our latent variable formulation can account for this uncertainty enabling the encoder to learn a linearized representation even in this setting.

6.4.1 Shallow Architecture Trained on Natural Data

To gain an intuition for the features learned by a phase-pooling architecture let us consider an encoder architecture comprised of the following stages: convolutional filter bank, rectifying point-wise nonlinearity, and phase-pooling. The decoder architecture is comprised of an un-pooling stage followed by a convolutional filter bank. This architecture was trained on simulated 32×32 movie frames taken from YouTube videos [15]. Each frame triplet is generated by transforming still frames with a sequence of three rigid transformations (translation, scale, rotation). More specifically let A be a random rigid transformation parameterized by τ, and let x denote a still image reshaped into a column vector, the generated triplet of frames is given by $\{f_1 = A_{\tau=\frac{1}{3}}x, f_2 = A_{\tau=\frac{2}{3}}x, f_3 = A_{\tau=1}x\}$. Two variants of this architecture were trained, their full architecture is summarized in the first two lines of Table 6.1. In Shallow Architecture 1, phase pooling is performed spatially in non-overlapping groups of 4×4 and across features in a one-dimensional topology consisting of non-overlapping groups of four. Each of the 16 pool-groups produce a code consisting of a scalar m and a three-component $\mathbf{p} = [p_f, p_x, p_y]^T$ (corresponding to two spatial and one feature dimensions); thus the encoder architecture produces a code of size $16 \times 4 \times 8 \times 8$ for each frame. The corresponding filters whose activations were pooled together are laid out horizontally in groups of four in Figure 6.3(a). Note that each group learns to exhibit a strong ordering corresponding to the linearized variable p_f. Because global rigid transformations can be locally well approximated by translations, the features learn to parameterize local translations. In effect the network learns to linearize the input by tracking common features in the video sequence. Unlike the spatial phase variables, p_f can linearize sub-pixel translations. Next, the architecture described in column 2 of Table 6.1 was trained on natural movie patches with the natural motion present in the real videos. The architecture differs in only in that pooling across features is done with overlap (groups of 4, stride of 2). The resulting decoder filters are displayed in Figure 6.3 (b). Note that pooling with overlap introduces smoother transitions between the pool groups. Although some groups still capture translations, more complex transformations are learned from natural movies.

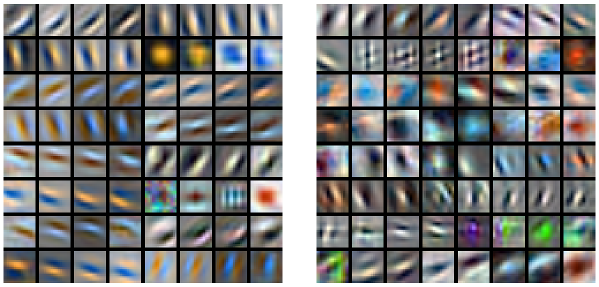
6.4.2 Deep Architecture trained on NORB

In the next set of experiments we trained deep feature hierarchies that have the capacity to linearize a richer class of transformations. To evaluate the properties of the learned features in a controlled setting, the networks were trained on simulated videos generated using the NORB dataset rescaled to 32×32 to reduce training time. The simulated videos are generated by tracing constant speed trajectories with random starting points in the two-dimensional latent space of pitch and azimuth rotations. In other words, the models are trained on triplets of frames ordered by their rotation angles. As

	Encoder	Prediction	Decoder
Shallow Architecture 1	Conv+ReLU $64 \times 9 \times 9$ Phase Pool 4	Average Mag. Linear Extrap. Phase	Conv $64 \times 9 \times 9$
Shallow Architecture 2	Conv+ReLU $64 \times 9 \times 9$ Phase Pool 4 stride 2	Average Mag. Linear Extrap. Phase	Conv $64 \times 9 \times 9$
Deep Architecture 1	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096	None	FC+ReLU $\mathbf{8192} \times 8192$ Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$
Deep Architecture 2	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096	Linear Extrapolation	FC+ReLU 4096×8192 Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$
Deep Architecture 3	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096 Reshape $64 \times 8 \times 8$ Phase Pool 8×8	Average Mag. Linear Extrap. Phase	Unpool 8×8 FC+ReLU 4096×8192 Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$

Table 6.1: Summary of architectures

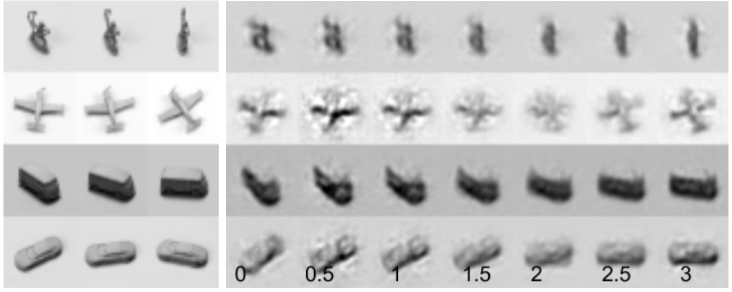
before, presented with two frames as input, the models are trained to predict the third frame. Recall that *prediction is merely a proxy for learning linearized feature representations*. One way to evaluate the linearization properties of the learned features is to linearly interpolate (or extrapolate) new codes and visualize the corresponding images via forward propagation through the decoder. This simultaneously tests the encoder's capability to linearize the input and the decoder's (generative) capability to synthesize images from the linearized codes. In order to perform these tests we must have an *explicit* code representation, which is not always available. For instance, consider a simple scheme in which a generic deep network is trained to predict the third frame from the concatenated input of two previous frames. Such a network does not even provide an explicit feature representation for evaluation. A simple baseline architecture that affords this type of evaluation is a Siamese encoder followed by a decoder, this exactly cor-



(a) Shallow Architecture 1

(b) Shallow Architecture 2

Figure 6.3: Decoder filters learned by shallow phase-pooling architectures



(a)

(b)

Figure 6.4: (a) Test samples input to the network (b) Linear interpolation in code space learned by our Siamese-encoder network

responds to our proposed architecture with the linear prediction layer removed. Such an architecture is equivalent to *learning the weights of the linear prediction layer* of the model shown in Figure 6.2. In the following experiment we evaluate the effects of: (1) fixing v.s. learning the linear prediction operator, (2) including the phase pooling operation, (3) including explicit curvature regularization (second term in Equation 6.1).

Let us first consider Deep Architecture 1 summarized in Table 6.1. In this architecture a Siamese encoder produces a code of size 4096 for each frame. The codes corresponding to the two frames are concatenated together and propagated to the decoder. In this architecture the first linear layer of the decoder can be interpreted as a learned linear prediction layer. Figure 6.4a shows three frames from the test set corresponding to temporal indices 1,2, and 3, respectively. Figure 6.4b shows the generated frames corresponding to interpolated codes at temporal indices: $0, 0.5, 1, 1.5, 2, 2.5, 3$. The images were generated by propagating the corresponding codes through the decoder. Codes corresponding to non-integer temporal indices were obtained by linearly interpolating in code space.

Deep Architecture 2 differs from Deep Architecture 1 in that it generates the predicted code via a fixed linear extrapolation in code space. The extrapolated code is then fed to the decoder that generates the predicted image. Note that the fully connected stage of the decoder has half as many free parameters compared to the previous architecture. This architecture is further restricted by propagating only the predicted code to the decoder. For instance, unlike in Deep Architecture 1, the decoder cannot copy any of the input frames to the output. The generated images corresponding to this architecture are shown in Figure 6.5a. These images more closely resemble images from the dataset. Furthermore, Deep Architecture 2 achieves a lower L^2 prediction error than Deep Architecture 1.

Finally, Deep Architecture 3 uses phase-pooling in the encoder, and “un-pooling” in the decoder. This architecture makes use of phase-pooling in a two-dimensional feature space arranged on an 8×8 grid. The pooling is done in a single group over all the fully-connected features producing a feature vector of dimension 192 (64×3) compared to 4096 in previous architectures. Nevertheless this architecture achieves the best overall L^2 prediction error and generates the most visually realistic images (Figure 6.5b).

6.4.3 Uncertainty

In this subsection we compare the representation learned by minimizing the loss in Equation 6.1 to Equation 6.7. Uncertainty is simulated by generating triplet se-

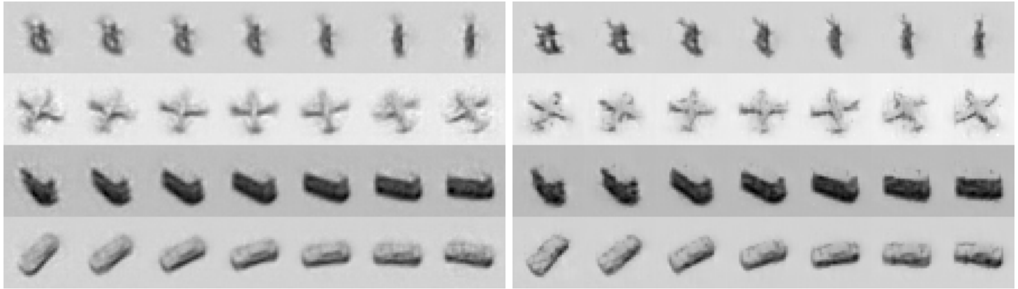


Figure 6.5: Linear interpolation in code space learned by our model. (a) no phase-pooling, no curvature regularization, (b) with phase pooling and curvature regularization

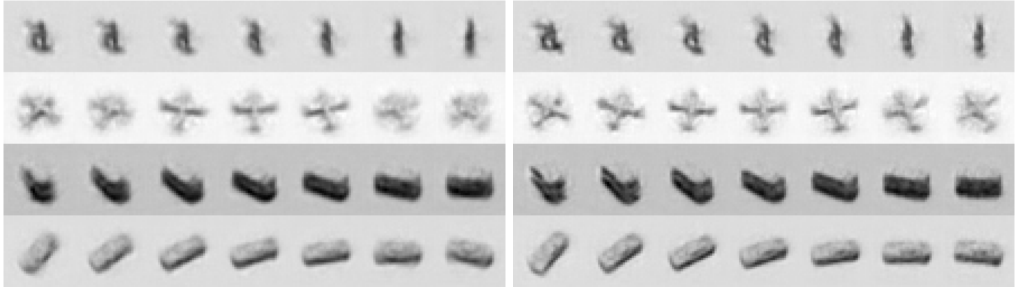


Figure 6.6: Interpolation results obtained by minimizing (a) Equation 6.1 and (b) Equation 6.7 trained with only partially predictable simulated video

quences where the third frame is skipped randomly with equal probability, determined by Bernoulli variable s. For example, the sequences corresponding to models with rotation angles $0°, 20°, 40°$ and $0°, 20°, 60°$ are equally likely. Minimizing Equation 6.1 with Deep Architecture 3 results in the images displayed in Figure 6.6a. The interpolations are blurred due to the averaging effect discussed in Subsection 6.3.2. On the other hand minimizing Equation 6.7 (Figure 6.6) partially recovers the sharpness of Figure 6.5b. For this experiment, we used a three-dimensional, real valued δ. Moreover training a linear predictor to infer binary variable s from δ (after training) results in a 94% test set accuracy. This suggests that δ does indeed capture the uncertainty in the data.

Chapter 7

Conclusion

Bibliography

[1] Guillaume Alain and Yoshua Bengio. “What regularized auto-encoders learn from the data-generating distribution”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3563–3593.

[2] Amir Beck and Marc Teboulle. “A fast iterative shrinkage-thresholding algorithm for linear inverse problems”. In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.

[3] Yoshua Bengio, Aaron Courville, and Pierre Vincent. “Representation learning: A review and new perspectives”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.8 (2013), pp. 1798–1828.

[4] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. *Representation Learning: A Review and New Perspectives*. Tech. rep. University of Montreal, 2012.

[5] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. “Signature verification using a Siamese time delay neural network”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 7.04 (1993), pp. 669–688.

[6] Joan Bruna and Stéphane Mallat. “Invariant scattering convolution networks”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.8 (2013), pp. 1872–1886.

[7] Joan Bruna, Arthur Szlam, and Yann LeCun. “Signal Recovery from Pooling Representations”. In: *ICML*. 2014.

[8] Charles F. Cadieu and Bruno A. Olshausen. “Learning Intermediate-Level Representations of Form and Motion from Natural Movies”. In: *Neural Computation* (2012).

[9] Taco S Cohen and Max Welling. “Transformation Properties of Learned Visual Representations”. In: *arXiv preprint arXiv:1412.7659* (2014).

[10] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[11] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[12] Daniel J Felleman and David C Van Essen. “Distributed hierarchical processing in the primate cerebral cortex”. In: *Cerebral cortex* 1.1 (1991), pp. 1–47.

[13] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[14] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. “Maxout Networks”. In: *ICML*. 2013.

[15] Ross Goroshin, Joan Bruna, Jonathan Tompson, David Eigen, and Yann LeCun. “Unsupervised Learning of Spatiotemporally Coherent Metrics”. In: *arXiv preprint arXiv:1412.6056* (2014).

[16] Ross Goroshin, Michael Mathieu, and Yann LeCun. “Learning to Linearize Under Uncertainty”. In: *arXiv preprint arXiv:1506.03011* (2015).

[17] Rostislav Goroshin and Yann LeCun. “Saturating Auto-Encoders”. In: *ICLR*. 2013.

[18] Karol Gregor and Yann LeCun. “Learning fast approximations of sparse coding”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 399–406.

[19] Raia Hadsell, Soumit Chopra, and Yann LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *CVPR*. 2006.

[20] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. “Transforming auto-encoders”. In: *Artificial Neural Networks and Machine Learning–ICANN 2011*. Springer, 2011, pp. 44–51.

[21] David H Hubel and Torsten N Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. In: *The Journal of physiology* 195.1 (1968), pp. 215–243.

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

[23] Aapo Hyvärinen, Jarmo Hurri, and Jaakko Väyrynen. “Bubbles: a unifying framework for low-level statistical properties of natural image sequences”. In: *JOSA A* 20.7 (2003), pp. 1237–1252.

[24] Koray Kavukcuoglu, MarcAurelio Ranzato, Rob Fergus, and Yann LeCun. “Learning Invariant Features through Topographic Filter Maps”. In: *CVPR*. 2009.

[25] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Micha´el Mathieu, and Yann LeCun. “Learning Convolutional Feature Hierarchies for Visual Recognition”. In: *NIPS*. 2010.

[26] Christoph Kayser, Wolfgang Einhauser, Olaf Dummer, Peter Konig, and Konrad Kding. “Extracting Slow Subspaces from Natural Videos Leads to Complex Cells”. In: *ICANN’2001*. 2001.

[27] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". MA thesis. University of Toronto, 2009.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *NIPS*. Vol. 1. 2. 2012, p. 4.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proc. IEEE* 86.11 (1998), pp. 2278–2324.

[30] Jorn-Philipp Lies, Ralf M Hafner, and Matthias Bethge. "Slowness and Sparseness Have Diverging Effects on Complex Cell Learning". In: 10 (3 2014).

[31] David G Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[32] Hossein Mobahi, Ronan Collobert, and Jason Weston. "Deep Learning from Temporal Coherence in Video". In: *ICML*. 2009.

[33] Aude Oliva and Antonio Torralba. "Modeling the shape of the scene: A holistic representation of the spatial envelope". In: *International journal of computer vision* 42.3 (2001), pp. 145–175.

[34] Bruno A Olshausen and David J Field. "Sparse coding of sensory inputs". In: *Current opinion in neurobiology* 14.4 (2004), pp. 481–487.

[35] Bruno A Olshausen and David J Field. "Sparse coding with an overcomplete basis set: A strategy employed by V1?" In: *Vision research* 37.23 (1997), pp. 3311–3325.

[36] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE. 2014, pp. 1717–1724.

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

[38] M Ranzato, Fu Jie Huang, Y-L Boureau, and Yann LeCun. “Unsupervised learning of invariant feature hierarchies with applications to object recognition”. In: *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[39] M.A Ranzato, Y-Lan Boureau, and Yann LeCun. “Sparse feature learning for deep belief networks”. In: *Advances in neural information processing systems* 20 (2007), pp. 1185–1192.

[40] Marc Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. “Unsupervised learning of invariant feature hierarchies with applications to object recognition”. In: *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[41] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. “Video (language) modeling: a baseline for generative models of natural videos”. In: *arXiv preprint arXiv:1412.6604* (2014).

[42] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. “Contractive auto-encoders: Explicit invariance during feature extraction”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 833–840.

[43] Sam T Roweis and Lawrence K Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *Science* 290.5500 (2000), pp. 2323–2326.

[44] Eero P Simoncelli and Bruno A Olshausen. “Natural image statistics and neural representation”. In: *Annual review of neuroscience* 24.1 (2001), pp. 1193–1216.

[45] Joshua B Tenenbaum, Vin De Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323.

[46] 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*. ACM. 2008, pp. 1096–1103.

[47] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. "Anticipating the future by watching unlabeled video". In: *arXiv preprint arXiv:1504.08023* (2015).

[48] Xiaolong Wang and Abhinav Gupta. "Unsupervised Learning of Visual Representations using Videos". In: *arXiv preprint arXiv:1505.00687* (2015).

[49] Laurenz Wiskott and Terrence J. Sejnowski. "Slow Feature Analysis: Unsupervised Learning of Invariances". In: *Neural Computation* (2002).

[50] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. "Performance-optimized hierarchical models predict neural responses in higher visual cortex". In: *Proceedings of the National Academy of Sciences* 111.23 (2014), pp. 8619–8624.

[51] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. 2014, pp. 3320–3328.

[52] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Robert Fergus. "Deconvolutional networks". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 2528–2535.

[53] Will Zou, Shenghuo Zhu, Kai Yu, and Andrew Y Ng. "Deep learning of invariant features via simulated fixations in video". In: *Advances in Neural Information Processing Systems*. 2012, pp. 3212–3220.