Gradient-based learning of higher-order image features

Roland Memisevic
Department of Computer Science
University of Frankfurt
ro@cs.uni-frankfurt.de

Abstract

Recent work on unsupervised feature learning has shown that learning on polynomial expansions of input patches, such as on pair-wise products of pixel intensities, can improve the performance of feature learners and extend their applicability to spatio-temporal problems, such as human action recognition or learning of image transformations. Learning of such higher order features, however, has been much more difficult than standard dictionary learning, because of the high dimensionality and because standard learning criteria are not applicable. Here, we show how one can cast the problem of learning higher-order features as the problem of learning a parametric family of manifolds. This allows us to apply a variant of a de-noising auto-encoder network to learn higher-order features using simple gradient based optimization. Our experiments show that the approach can outperform existing higher-order models, while training and inference are exact, fast, and simple.

1. Introduction

Unsupervised feature learning has become an area of major interest in many vision tasks that involve some form of recognition (including recognition of objects, scenes, actions). Feature learners are typically defined as two-layer networks, or graphs, that connect a layer of pixels with a layer of "hidden units". Each hidden unit defines a linear projection, or "filter", and the set of all filter responses defines a parsimonious code for the observation (for example, [21], [23], [17], [22], [1]. Most common learning criteria are reconstruction error or approximate probability of the observed images. When applied to natural image patches, Gabor-filters emerge naturally in practically all the different methods. Besides their biological plausibility, feature learners tend to perform very well in recognition tasks.

An extension of feature learning that has received a lot of attention recently, is the learning of *relations* between pixel intensities, rather than of pixel intensities themselves [18], [12], [25], [19]. For this end, one can extend the bi-partite graph of a standard sparse coding model with a *tri-partite* graph that connects hidden variables with *two* images. Hidden units then turn into "mapping" units that model structure in the relationship *between* two images rather than static structure *within* a single image. The model as a whole thus can be thought of as a model of image transformations [18]. This type of model has recently been applied to learning a representation of optical flow [18], spatio-temporal features [30], class-specific transformations [11], [29], or transparent motion [19]. Learning relations between pixels is closely related to bi-linear modeling [26], [20], [31], and it has been extended to mixed data-types, such as image/eye position [15].

Recently, the special case of relational feature learning, where the two images are *the same*, has attracted some interest. In this case, hidden units encode within-image pixel covariances or, equivalently, higher-order image features [12], [25], [24], [4]. An instantiation of the vector of hidden variables in this type of model can be thought of as encoding a conditional covariance matrix over pixels, instead of a conditional mean, as would be the case in a standard sparse coding model. Learning of feature covariances was shown to yield good results in a variety of recognition and classification tasks (for example, [24], [5]). An alternative interpretation for why the inclusion of higher-order features works well is, that they are better at representing real-valued (and, in particular, heavy-tailed) data [24], [4].

Although higher-order features can improve the performance in many recognition tasks, learning them has been considerably harder than training standard sparse coding models. The reason is that learning of higher-order features amounts to learning on a basis-expansion of the inputs, such as on all pair-wise products of pixel intensities. As a result, the dimensionality and the number of parameters grow at least quadratically with the number of input pixels. A common solution is to pre-project input images onto basis-functions *before* computing pair-wise products [19], [25], [12]. Unfortunately, for learning one has to invert these projections in order to compute objective functions and gradients. Naive training thus leads to a computational complex-

ity that remains quadratic in the number of input components. More importantly, existing methods have to rely on sampling-based schemes, such as Hybrid Monte Carlo [24] or various modifications of contrastive divergence learning ([9]) to deal with the presence of three-way cliques [29].

In this work we cast the problem of learning higher-order features as the task of learning *a family of manifolds*. This makes it possible to learn higher-order features using a "relational" variant of an auto-encoder. The model is similar to the classical auto-encoder (for example, [10]), with the difference that parameters are *gated* by additional inputs, which turns the auto-encoder into a type of a higher-order neural network [7]. When using a single hidden layer with linear activations, the model takes the form of a (conditionally trained) *bi-linear model* [31] [8], [20], where additional "simple cells" can be used to pre-project inputs before computing multiplicative interactions. Training both relational and within-image higher-order features is possible with simple back-propagation. Potential advantages of this model are that (a) low-dimensional pre-projections or multi-layer versions of the model can be defined naturally, (b) by using back-propagation, it is not necessary to manually calculate gradients, and one can use modern code-generation methods to transparently parallelize code (for example, [2]), (c) the model makes no difference between learning of covariance-features and learning of transformations, (d) covariance-features can be mixed with standard features by simply adding connections that are not gated, (e) in order to deal with binary, real-valued and other types of observables one can simply use the appropriate activation/cost-functions in the final layer of the network, such as squared error for real-valued data and log-loss for binary data.

Computing three-way-products of low-dimensional projections has been known as "pi-sigma"-learning in the literature [28]. Higher-order feed-forward networks themselves date back to at least to the 1980's [7]. Neither higher-order projections nor simple higher-order networks have been applied in an auto-encoder or for feature learning to the best of our knowledge.

2. Learning a family of manifolds

2.1. Auto-encoders and de-noising auto-encoders

Let $\boldsymbol{y} \in I\!R^{n_Y}$ be a high-dimensional input vector with components $y_1, \ldots, y_{n_Y}$. An auto-encoder defines a mapping $\hat{\boldsymbol{y}}(\boldsymbol{h}(\boldsymbol{y}))$, consisting of an *encoder* $\boldsymbol{h}(\cdot)$ and a *decoder* $\hat{\boldsymbol{y}}(\cdot)$ that is trained to reconstruct the input data. The encoder is usually defined as a nonlinear function applied to a linear projection of $\boldsymbol{y}$:

$$h_k(\boldsymbol{y}) = \sigma\big(\sum_j w_{kj} y_j\big) \tag{1}$$

with $\sigma(\cdot)$ being an element-wise sigmoid activation function, such as $\sigma(a) = (1 + \exp(-a))^{-1}$. The decoder reconstructs $\boldsymbol{y}$ from the hidden units[1]:

$$\hat{y}_j(\boldsymbol{h}(\boldsymbol{y})) = \sum_k w_{kj} h_k(\boldsymbol{y}). \tag{2}$$

Traditionally, n_H, the number of components of $\boldsymbol{h}(\boldsymbol{y})$, is set to be smaller than the dimensionality of $\boldsymbol{y}$, such that it constitutes a bottleneck, which forces the model to compress the input before reconstructing it [10]. Recently, [34] showed that instead of a bottleneck layer one may use an *overcomplete* hidden layer, if one *corrupts* the inputs before computing hidden layer activities. Since reconstruction-targets are left unchanged, the model then effectively learns to de-noise the data. This model has been referred to as a "de-noising auto-encoder", and it was shown to outperform not only standard auto-encoders, but also a variety of other sparse coding methods on recognition tasks [34].

Several options exist for the type of noise-process to use. A simple and effective one is *blanking out random components* of the input vector [34]. For this end, a random fraction, η, of input-pixels is chosen randomly in each training iteration and multiplied by zero. The task of the model thereby becomes to fill in the components thus turned into "missing entries". Optimal values for η are typically in the range from 0.1 to 0.5. An important feature of de-noising auto-encoders is that they make back-propagation applicable to sparse coding, since they allow for learning of overcomplete representations using gradient-descent.

2.2. Relational auto-encoders

When the appearance of an object undergoes transformations due to lighting-, pose-, or other variations, images of the object may be thought of as tracing out a manifold[2] in the high-dimensional pixel-space. The intrinsic dimensionality of this manifold is equal to the number of degrees of freedom in the transformations.

When dealing with *image pairs* ($\boldsymbol{x} \in I\!R^{n_X}$, $\boldsymbol{y} \in I\!R^{n_Y}$), which are related through transformations, one can think of the outputs $\boldsymbol{y}$ as being confined to a *conditional* appearance manifold. In contrast to the static case this manifold is a function of the input image $\boldsymbol{x}$, in that different inputs $\boldsymbol{x}$ give rise to different manifolds over $\boldsymbol{y}$. If we assume that this dependency is smooth, so that small variations in the input image $\boldsymbol{x}$ lead to small variations in the manifold associated

[1] We use an auto-encoder with tied parameters here for convenience, where decoder parameters are the transpose of encoder parameters. But one can also use untied weights.

[2] While the manifold-metaphor is useful to derive mathematical arguments, real-world data can be structured in a form that is different from a densely sampled, homogeneous manifold. When using overcomplete codes, a more accurate requirement is that the entropy of the data-distribution is smaller than the entropy of a uniform distribution in data-space.

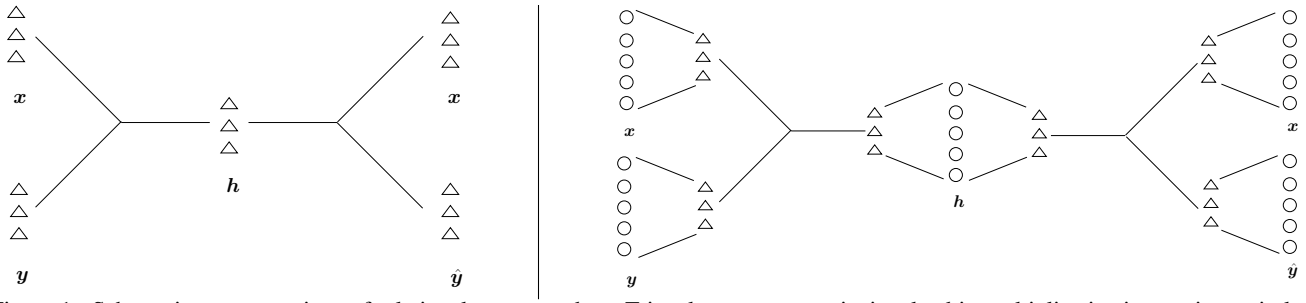

Figure 1. Schematic representations of relational auto-encoders. Triangles represent units involved in multiplicative interactions, circles represent standard units. **Left:** Naive version. Hidden unit activations are computed by combining inputs $\boldsymbol{x}$ multiplicatively with inputs $\boldsymbol{y}$; output activations by combining hidden units multiplicatively with inputs $\boldsymbol{x}$. **Right:** Model with additional intermediate layers.

with $\boldsymbol{x}$, then we can define the problem of learning about transformations as learning a **family of manifolds, parameterized by the input image $\boldsymbol{x}$.**

For this end we now turn to the auto-encoder network as a simple parametric manifold learning method. We can define an auto-encoder as a *conditional* model of $\boldsymbol{y}$ given $\boldsymbol{x}$, if we let the model parameters w_{kj} be a function of $\boldsymbol{x}$. The role of $\boldsymbol{x}$, then, is to modulate the parameters of the model as suggested. To let w_{kj} be a linear function of $\boldsymbol{x}$, we define

$$w_{kj}(\boldsymbol{x}) = \sum_i \hat{w}^i_{kj} x_i \tag{3}$$

where $\hat{w}^i_{kj}$ are linear parameters associated with parameter w_{kj} of the auto-encoder. Plugging Eq. 3 into Eq. 1 shows that in the conditional model, hidden variable activities are now given by a simple basis expansion of $\boldsymbol{x}$ and $\boldsymbol{y}$

$$h_k(\boldsymbol{x}; \boldsymbol{y}) = \sigma\big(\sum_{ij} \hat{w}^i_{kj} x_i y_j\big), \tag{4}$$

and the outputs $\hat{\boldsymbol{y}}$ are given by a basis expansion of $\boldsymbol{x}$ and $\boldsymbol{h}$

$$\hat{y}_j(\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{y})) = \sum_{ki} \hat{w}^i_{kj} x_i h_k(\boldsymbol{x}, \boldsymbol{y}). \tag{5}$$

An illustration of the model is depicted in Figure 1 (left plot). The network takes two data-cases, $\boldsymbol{x}$ and $\boldsymbol{y}$, as input, and it uses pair-wise products between the components of $\boldsymbol{x}$ and $\boldsymbol{y}$ as inputs to the hidden variables. Feeding pair-wise products into the hidden units turns these into *relational* or *"cross-correlation"-encoders*, that learn to represent patterns of co-occurrence among the components of $\boldsymbol{x}$ and $\boldsymbol{y}$. While hidden unit activities simply rely on a basis-expansion of the inputs, it is important to note that reconstructions for $\boldsymbol{y}$ are computed similarly, by multiplicatively combining the vector of $\boldsymbol{h}$ with the known vector $\boldsymbol{x}$. As a result, we can deploy standard learning criteria, such as reconstruction error to train the three-way parameters. In particular, for real-valued data $\boldsymbol{y}$, we can minimize

$$E = \sum_\alpha \big(\hat{\boldsymbol{y}}(\boldsymbol{h}(\boldsymbol{x}^\alpha; \boldsymbol{y}^\alpha)) - \boldsymbol{y}^\alpha\big)^2 \tag{6}$$

using a set of training pairs $\{(\boldsymbol{x}^\alpha, \boldsymbol{y}^\alpha)\}$. For binary or multinomial $\boldsymbol{y}$, we minimize cross-entropy loss (negative log-probability). As with a standard auto-encoder, one can use back-propagation to compute gradients and use gradient-based optimization for learning. The de-noising criterion (cf. Section 2.1) applies without change: We simply need to corrupt $\boldsymbol{x}$ and $\boldsymbol{y}$ independently.

The model defines a "*conditional manifold*" over $\boldsymbol{y}$ as a function $\boldsymbol{x}$. This is in contrast to [18], for example, who define a conditional *distribution*. The model is an instance of a higher-order neural network, i.e. a network whose units compute products of incoming variables, not just weighted sums [7]. We refrained from adding "bias terms" when defining hidden unit activities and output activities to avoid clutter, although in practice it is advisable to define the net input of each units as an affine, rather than a linear, function. Alternatively, one can think of data and hidden variables as being in homogeneous coordinates, i.e. with an extra, constant "1"-dimension.

2.3. Adding "simple cells"

The number of three-way parameters in the model described thus far is $(n_X \times n_Y \times n_H)$ (Eq. 3). This can be prohibitively large in practice, in particular, if one assumes the number of relational units to be roughly on the same order as the number of inputs and outputs.

To reduce the large number of parameters, it can be useful to impose restrictions on the parameter "tensor" $W = \big(\hat{w}^i_{kj}\big)_{ijk}$. One such restriction is to allow three-way connections only for subsets of variable triplets, rather than for all. If we assume the number of input, output and relational units to be *the same*, we can restrict the parameter tensor, such that every relational unit is connected to exactly one input unit and to exactly one output unit [19].

This might seem overly restrictive at first sight. Also, the assumption of equal numbers of units may not often hold in practice. Note, however, that we can add an intermediate representation, which can *learn* to deal with these restrictions optimally. For this end, we can *project* $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{h}$ onto matrices (or "latent factors") F^X, F^Y and F^H,

respectively, and allow for element-wise (or in other ways restricted) three-way interactions in these projections. For element-wise interactions, the dimensionality, n_F, of the latent factors needs to be identical for inputs, outputs and hidden units. In this case, F^X is an $n_X \times n_F$, F^Y is an $n_Y \times n_F$ and F^H is an $n_H \times n_F$ matrix. Since we can absorb element-wise three-way interactions into the factor matrices, we can let these constitute the whole set of parameters. By using symmetry between the encoder and decoder networks, we can then define hidden unit activities as

$$h_k(\boldsymbol{x};\boldsymbol{y}) = \sigma\big(\sum_f F^H_{kf} \sum_i F^X_{if} x_i \sum_j F^Y_{jf} y_j\big). \quad (7)$$

Output activities, given hidden units and inputs $\boldsymbol{x}$, are computed similarly as

$$\hat{y}_j\big(\boldsymbol{h}(\boldsymbol{x},\boldsymbol{y})\big) = \sum_f F^Y_{jf} \sum_i F^X_{if} x_i \sum_k F^H_{kf} h_k(\boldsymbol{x};\boldsymbol{y}) \quad (8)$$

The overall architecture is shown in Figure 1 (right) plot. It is possible to extend the model to define larger architectures where high-order units are mixed with standard hidden units. It is also possible to use different filters in the encoder and the decoder-network.

3. Mean-covariance auto-encoders

The sole function of the inputs, $\boldsymbol{x}$, in the definition of the conditional auto-encoder is to modulate parameters (Eq. 3). This is why training – despite the presence of three-way interactions – amounts to simple gradient-based optimization. Nothing in the definition of the model, however, requires that $\boldsymbol{x}$ is the input and $\boldsymbol{y}$ is the output. The two types of argument are interchangeable. In practice, as we shall show, it can be useful to train the model symmetrically, by reconstructing both $\boldsymbol{y}$ from $\boldsymbol{x}$ and $\boldsymbol{x}$ from $\boldsymbol{y}$. A simple way to achieve this is by defining the overall objective function as the sum of the two asymmetric objectives

$$E = \sum_\alpha \big(\hat{\boldsymbol{y}}\big(\boldsymbol{h}(\boldsymbol{x}^\alpha;\boldsymbol{y}^\alpha)\big) - \boldsymbol{y}^\alpha\big)^2 + \sum_\alpha \big(\hat{\boldsymbol{x}}\big(\boldsymbol{h}(\boldsymbol{x}^\alpha;\boldsymbol{y}^\alpha)\big) - \boldsymbol{x}^\alpha\big)^2 \quad (9)$$

Using the symmetric objective can be thought of as the non-probabilistic analog of modeling a *joint* distribution over $\boldsymbol{x}$ and $\boldsymbol{y}$ as opposed to modeling a conditional.

3.1. Setting $\boldsymbol{x} = \boldsymbol{y}$

As an important special case of symmetric training, we now consider the case where input and output image are the same. As we discussed in Section 2.1, de-noising auto-encoders can prevent an overcomplete hidden layer from learning the trivial identity mapping. Here, we suggest using the same approach to prevent relational hidden variables from learning the identity in a *covariance model of a single image*.

When training on pairs of identical images, higher-order hidden variables can encode *within-image* covariance structure [25], [12]). We can use a relational auto-encoder to model within-image covariances by setting $x = y$ and using the de-noising criterion exactly as before (cf. Section 2.2). When dealing with two identical copies of a single input image $\boldsymbol{x}$, we apply the noise process twice, and independently, to each copy. We can thus train the model in the same way as a standard relational auto-encoder.

It is interesting to note that, while a de-noising auto-encoder makes use of *one* perturbation of the input in order to model pixel-"means", the relational auto-encoder utilizes *two* differently perturbed copies of the same stimulus, which enables it to model covariances. In other words, **in order to encode covariance patterns, the model compares multiple noisy copies of the same input**. Interestingly, the only piece of biological hardware required to implement this scheme are multiple feature detectors that encode the same feature, and an architecture where these multiple encodings feed into the same pooling unit.

[25] suggest using various positivity constraints on the model parameters to encourage positive-definiteness of the covariance-matrices implicitly defined by that model. With the auto-encoder, positivity constraints can be enforced, if desired, by using a change of variables, such as $F_{if} \leftarrow \exp(F_{if})$, still allowing for training with unconstrained gradient descent. But we found in our experiments, that it is usually possible to train the model without such constraints or changes of variables. We did find, however, that one can obtain better stability by re-normalizing filters throughout the optimization, such that filters grow slowly and maintain roughly the same relative size. In analogy to probabilistic models, we shall refer to a relational auto-encoder with only covariance-units as "covariance auto-encoder" (cAE) and a model that also includes standard hidden units as "mean-covariance auto-encoder" (mcAE). Furthermore, we shall refer to hidden units encoding higher-order structure within an image as "covariance-units", and those encoding across-image structure as "relational units" or "cross-correlation" units.

4. Experiments

4.1. Learning image transformations

In a first set of experiments[3] we trained the relational auto-encoder on pairs of transformed image patches, forcing hidden variables to encode the transformations [18], [19], [30].

First, we applied the model to pairs of shifted random dot images using the data-set provided on the accompanying website of [19]. The data-set consists of 10000 binary

[3] A Python/theano implementation of the model is available at http://www.cs.toronto.edu/~rfm/code/rae/index.html

image patch pairs of size 13×13 pixels each, where the output image in each pair is a translated version of the input image, and both translation direction and amount are drawn uniformly randomly. We trained a model with 100 filters and 25 relational hidden units using cross-entropy loss. F^X and F^Y are both of size 169×100, F^H is 25×100. We trained the model with stochastic gradient descent, using mini-batches of size 100 image pairs for about 1000 epochs. Filter pairs learned by the model are shown in Figure 2 (top row). The figure shows how the model, similar to [19], learns *phase-shifted Fourier components* to represent translations.

To compare the performance quantitatively against [19], we created an evaluation set by splitting the data-set into 5000 training, 2500 validation, and 2500 test cases. We performed a search over learning-rate, number of training-epochs, filter normalization, and, for the auto-encoder, corruption-level η. We computed cross-entropy-reconstruction error for both models on the validation and test set. The test set performance of the winners on the validation set is **33.29** for the auto-encoder and **34.42** for the model described in [19], showing that the auto-encoder can yield competitive performance. Interestingly, this result parallels recent work on classification, where de-noising auto-encoders tend to slightly outperform restricted Boltzmann machines [34].

We also trained the model on real-valued patches cropped from a subset of the Tiny Images-dataset consisting of 80 million images of size 32×32 pixels [32]. We cropped patches of size 16×16 pixels randomly from the images and transformed them (i) using random affine transformations and (ii) using artificial *motion discontinuities* (details below). Affine filters are depicted in Figure 2 (middle row). The filters are similar to rotation filters presented in [19].

We generated a data-set with motion discontinuities as follows. We sampled (uniformly randomly) a line separating a 10×10-patch, cropped from the Tiny Images-dataset, into two regions. We then sampled a translation direction and amount for each of the two regions separately and performed a separate translation in the two regions. We made sure that no empty space-artifacts were generated in this process. As a result the data-sets exhibits typical aspects of real-world motion-discontinuities, such as occlusion and disocclusion [3]. Because of the combinatorial explosion, the number of possible transformations is huge. We trained the relational auto-encoder using 250000 patch-pairs. One objective of this experiment is to investigate whether the auto-encoder is able to learn a *factorial* representation to deal with the combinatorial explosion. The learned filters are depicted in the bottom row of Figure 2. The figure shows that in order to learn motion discontinuities the model projects input images onto localized *Gabor pairs*. It is interesting to note that Gabor features are useful here for a slightly different reason than to represent natural images: Gabor filter pairs make it possible to de-compose the observed transformations into local shifts and thereby to deal with the combinatorial number of transformations.

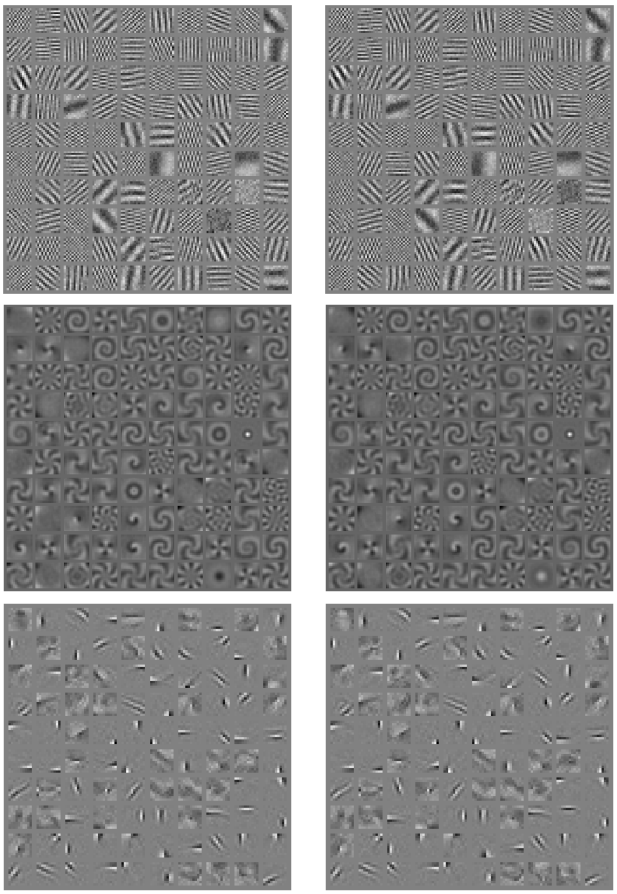

Figure 2. Filter pairs learned from image transformations: Translations (top), affine (middle), motion discontinuities (bottom). The left column shows input filters, the right column shows output filters.

4.2. Encoding motion with a "bag of warps"

Relational hidden units can model transformations by combining a set of factors which, in turn, multiplicatively gate the responses of filters between inputs and outputs (cf. Eq. 8 and Fig. 1). Each instantiation of hidden variables thereby represents a *warp*, that is, a transformation in the space of "stacked" gray-value intensities. We can extend the model to define a "*bag-of-warps*" by adding the responses of relational units at various positions in an image or in a video, with possible application, for example, in video classification.

To test, whether the model can learn meaningful structure from natural videos, we first trained a model with 400 filters and 400 relational hidden variables on adjacent time frames of size 16×16 pixels extracted from

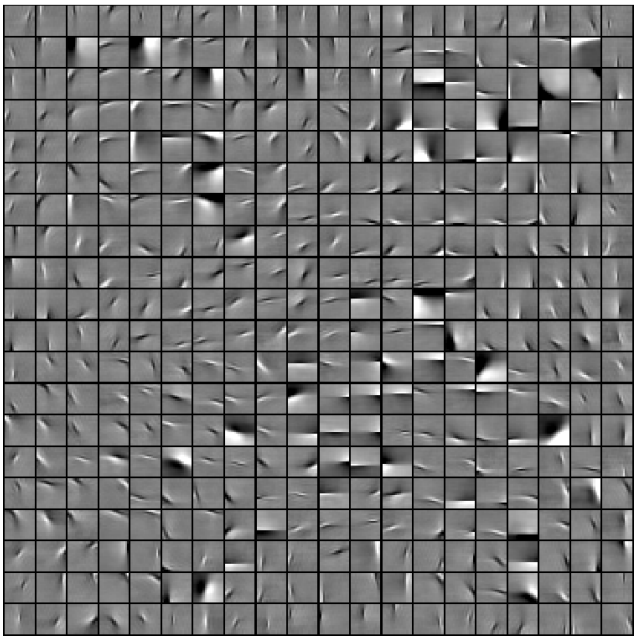

Figure 3. Input filters learned from broadcast television. (Corresponding output filters shown in Figure 4.)

the van Hateren broadcast television database [33]. We PCA-whitened the patches first, by projecting onto the first 171 PCA-components. We initialized the relational auto-encoder with a two-dimensional topology, by constraining the matrix F^H, such that nearby hiddens on a two-dimensional grid are connected to nearby factors (see, for example, [24]). Figures 3 and 4 show the learned input and output filters, respectively, and show that the model learns to represent natural video using phase-shifted, Gabor-like filters.

We then trained a "bag-of-warps" model with 1024 relational units and 1024 factors using the KTH training data-set [27], where we crop patch pairs from adjacent frames around SIFT-keypoints. The data-set consists of videos of 6 types of action performed by humans. We define the representation for a single video as the sum of relational hidden variables across all keypoints and all frames. Plugging this representation into a multinomial logistic regressor yields a test-set recognition rate of 80.56% correct, which compares reasonably well with existing methods, considering the simplicity of the approach, and the fact that keypoints are spatial not spatio-temporal.

4.3. Learning within-image correlations

To evaluate the model's ability to learn within-image structure, we trained a model on 2 million patches of size 8×8 pixels cropped from the Tiny Images-dataset. Figure 5 shows 576 learned filters. Like [24], we used a 1-dimensional topography to initialize the matrix F^H.

We used the CIFAR-10 dataset [13], a labelled subset of the Tiny Images-dataset to evaluate the model quantitatively. The CIFAR-10 dataset consists of 50000 training images and 10000 test images of size 32×32. There are 10 different object classes. For comparability, we performed a numerical evaluation following the same protocol as used by [24] and [25]. Specifically, we trained both a covariance encoder and a mean-covariance encoder on patches of size 8×8 pixels cropped from a subset of the TINY-dataset [32] which do not overlap with the CIFAR-10-dataset.

After training, we extracted features convolutionally by computing hidden unit activities on a regular 7×7 grid from the image. Like [24], we compared one model with 225 filters and 225 covariance variables, one model with 900 factors and 225 covariance variables, and one model with 576 factors (shown in Figure 5), 144 relational variables and 81 mean variables. For all models, the total number of hidden variables (including both covariance- and mean-encoding variables) is 225. The dimensionality of the convolutional feature vector is $49 \times 225 = 11025$ in all cases. We trained a multinomial logistic regression model for classification. We used no weight-decay for training the models and a weight-decay of 0.001 for the logistic regression model in all experiments. We did *not* vary hyperparameters such as the corruption level, which we set to 0.5 in all experiments. Performing cross-validation would allow us to set these and other hyper-parameters (such as weight-decay and learning-rate) differently for each run and therefore likely improve the performance. In contrast to [24], for we train all model-parameters at once, rather than holding any subsets of parameters fixed at any time.

Table 1 shows classification performance in $\%$-correct. The performance for our model is in general comparable to that of the mcRBM, where in 2 out of 3 cases the relational auto-encoder shows better performance. Like for similar probabilistic models, adding "mean"-units to the model improves performance. It is important to note, that training the auto-encoder is significantly simpler and faster than training the mcRBM using hybrid Monte Carlo [24]. In addition to the mcRBM, we also compared to a simple denoising auto-encoder with 225 hidden units, but we never achieved recognition rates beyond approximately 40% with that model, which seems to indicate that covariance-codes are key to good recognition rates for the setting (patchsize, number, features, etc.) that we used.

A careful search over parameters such as the input patch size, stride in the convolutional extraction or choice of top-level classifier, is likely to improve performance. While the model is likely to generalize to other settings and other patch sizes, the main purpose of this experiment is not to achieve the best possible object recognition rate per se, but to show that the relational auto-encoder provides an effective way to extract higher-order structure from images.

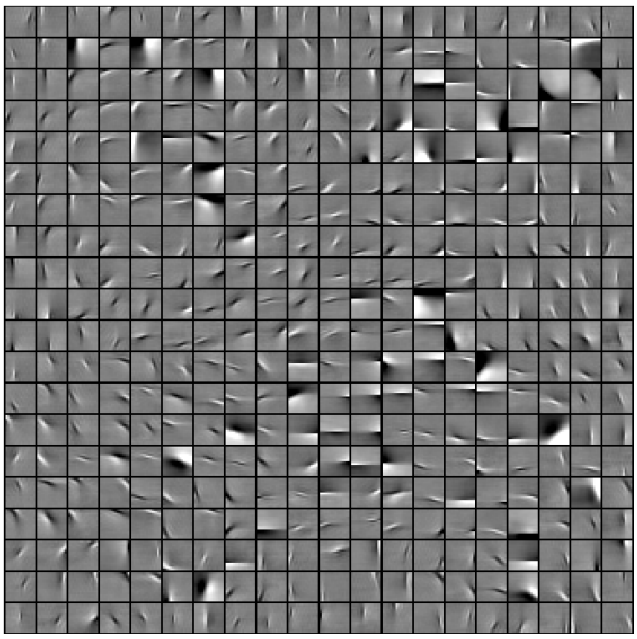

Figure 4. Output filters learned from broadcast television. (Corresponding input filters shown in Figure 3.)

#factors / #covar-hiddens / #mean-hiddens	Model	Perf (%)
225 / 225 / 0	cRBM	63.6
225 / 225 / 0	cAE	64.5
900 / 225 / 0	cRBM	64.7
900 / 225 / 0	cAE	65.4
576 / 144 / 81	mcRBM	68.2
576 / 144 / 81	mcAE	67.7

Table 1. CIFAR-10 recognition rates.

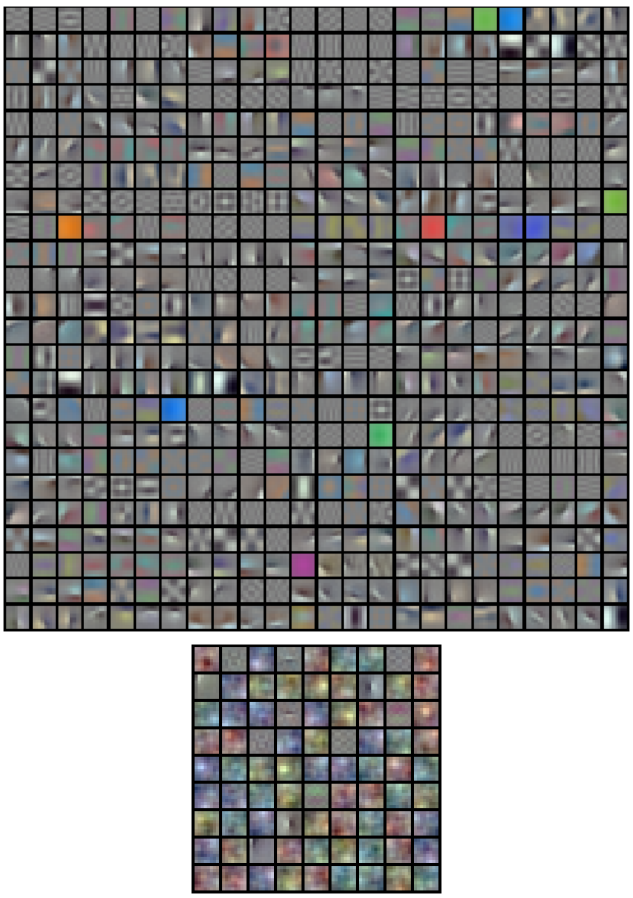

Figure 5. Within-image filters learned from a subset of the Tiny Images. **Top:** Covariance filters. **Bottom:** Mean filters. Best viewed in color.

5. Discussion

Correspondence is a ubiquitous and fundamental concept in computer vision, which is at the heart of problems like stereo, motion understanding, tracking, odometry and recognition. Relational sparse coding is a way to introduce *learning* into the modeling of correspondences. Relational hidden units could be thought of as "hubs", which *combine* information sources. Thus they can complement units which simply *encode* information. One can envision hierarchies containing both multiplicative "hubs" that combine information and feature-learners that encode information (including higher-order information learned by "hubs", such as depth and motion). Such an "*encoder/combiner*-network" could still be trained with simple back-propagation. Interestingly, computing relational hidden unit activities amounts to pooling over rectified filter-responses, so hidden units act like complex cells and factors like simple cells. The same is true in probabilistic models [19]. The idea that complex cells, either through multiplicative interactions, or equivalently through "squaring", can act as information combiners, dates back at least to energy and cross-correlation models (see, for example, [6] and references therein). But there has been little work on learning such models from data.

Relational auto-encoders bear some interesting connections to structured prediction [14]. High-dimensional prediction is hard, because of the combinatorial explosion that follows from taking output dependencies into account. One approach to dealing with this, is to ignore the dependencies among the output-components, and to predict each component independently, as would be the case, for example, with linear regression. Another approach is to assume restricted dependencies among the outputs, such as a chain- or tree-structure, in which case one can use dynamic programming or belief-propagation for inference and learning [16], [14]. Learning a conditional *manifold* can be viewed as a third way of predicting high-dimensional outputs, where, in contrast to conditional graphical models, we *learn* the conditional dependency structure with latent variables.

Acknowledgments

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the project 01GQ0841 (BFNT Frankfurt).

References

[1] A. J. Bell and T. J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision research*, 37(23):3327–3338, December 1997. 1

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference.*, 2010. 2

[3] M. J. Black and D. J. Fleet. Probabilistic detection and tracking of motion boundaries. *Int. J. Comput. Vision*, 38:231–245, July 2000. 5

[4] A. Courville, J. Bergstra, and Y. Bengio. A spike and slab restricted boltzmann machine. In *Artificial Intelligence and Statistics*, 2011. 1

[5] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In *Adv. in Neural Information Processing Systems 23*. 2010. 1

[6] D. Fleet, H. Wagner, and D. Heeger. Neural encoding of binocular disparity: Energy models, position shifts and phase shifts. *Vision Research*, 36(12):1839–1857, June 1996. 7

[7] C. L. Giles and T. Maxwell. Learning, invariance, and generalization in high-order neural networks. *Appl. Opt.*, 26(23):4972–4978, Dec 1987. 2, 3

[8] D. Grimes and R. Rao. Bilinear sparse coding for invariant vision. *Neural Computation*, 17(1):47–73, 2005. 2

[9] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:2002, 2000. 2

[10] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006. 2

[11] G. B. Huang and E. Learned-Miller. Learning class-specific image transformations with higher-order Boltzmann machines. In *In Workshop on Structured Models in Computer Vision at CVPR, 2010*, 2010. 1

[12] Y. Karklin and M. Lewicki. Emergence of complex cell properties by learning to generalize in natural scenes. *Nature*, 457:83–86, January 2009. 1, 4

[13] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Toronto, Canada, 2009. 6

[14] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2001. 7

[15] H. Larochelle and G. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *Adv. in Neural Information Processing Systems 23*. 2010. 1

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. 7

[17] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE transactions on image processing*, (1):53–69, January 2008. 1

[18] R. Memisevic and G. E. Hinton. Unsupervised learning of image transformations. In *CVPR 2007*, 2007. 1, 3, 4

[19] R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22(6):1473–92, 2010. 1, 3, 4, 5, 7

[20] B. Olshausen, C. Cadieu, J. Culpepper, and D. Warland. Bilinear models of natural images. In *SPIE Proceedings: Human Vision Electronic Imaging XII*, San Jose, 2007. 1, 2

[21] B. Olshausen and D. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 1996. 1

[22] S. Osindero, M. Welling, and G. E. Hinton. Topographic product models applied to natural scene statistics. *Neural Computation*, 18:381–414, February 2006. 1

[23] M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Adv. in Neural Information Processing Systems*, 2007. 1

[24] M. Ranzato and G. E. Hinton. Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines. In *CVPR*, 2010. 1, 2, 6

[25] M. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-Way Restricted Boltzmann Machines For Modeling Natural Images. In *Artificial Intelligence and Statistics*, 2010. 1, 4, 6

[26] R. Rao and D. Ballard. Efficient encoding of natural time varying images produces oriented space-time receptive fields. Technical report, Rochester, NY, USA, 1997. 1

[27] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: A local svm approach. In *ICPR'04*. IEEE Computer Society, 2004. 6

[28] Y. Shin and J. Ghosh. The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *in Proceedings of the International Joint Conference on Neural Networks*, 1991. 2

[29] J. Susskind, R. Memisevic, G. Hinton, and M. Pollefeys. Modeling the joint density of two images under a variety of transformations. In *CVPR 2011*, 2011. 1, 2

[30] G. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *Proc. European Conference on Computer Vision*, 2010. 1, 4

[31] J. Tenenbaum and W. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000. 1, 2

[32] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1958–1970, 2008. 5, 6

[33] L. van Hateren and J. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proc. Biological Sciences*, 265(1412):2315–2320, 1998. 6

[34] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*. 2, 5