Department of Computer Science
University of Toronto
http://learning.cs.toronto.edu

6 King's College Rd, Toronto
M5S 3G4, Canada
fax: +1 416 978 1455

December 31, 2006

UTML TR 2006–5

Unsupervised Learning of Image Transformations

Roland Memisevic and Geoffrey Hinton
Department of Computer Science, University of Toronto

Unsupervised learning of image transformations*

Roland Memisevic
University of Toronto
roland@cs.toronto.edu

Geoffrey Hinton
University of Toronto
hinton@cs.toronto.edu

December 31, 2006

Abstract

We describe a probabilistic model for learning rich, distributed representations of image transformations. The basic model is a gated conditional random field that is trained to predict transformations of its inputs using a factorial set of latent variables. Inference in the model consists in extracting the transformation, given a pair of images, and can be performed exactly and efficiently.

We show that, when trained on natural videos, the model develops domain specific motion features, in the form of fields of locally transformed edge filters. When trained on affine, or more general, transformations of still images, the model develops codes for these transformations, and can subsequently perform recognition tasks that are invariant under these transformations. It can also fantasize new transformations on previously unseen images. We describe several variations of the basic model and provide experimental results that demonstrate its applicability to a variety of tasks.

1 Introduction

Natural images are not random, but show a great deal of statistical regularity, both at the level of single pixels and at the level of larger regions. Unsupervised learning has been used to discover the statistical structure present in images from training data, and many unsupervised algorithms (such as PCA, ICA, and many others) are now an essential part of the standard toolbox for solving recognition, detection, denoising, and other tasks.

There has been little work on the related, but also more difficult problem, of discovering structure in the ways images *change*. Typical transformations of images in, say videos, are highly regular and structured, and systems can profit from discovering, and then exploiting this structure.

How images can be transformed is intricately related to how images themselves are structured: Natural images, that are composed of edges and junctions, etc., will

*This work was first presented at the Canadian Institute for Advanced Research Summer School on Vision and Learning, Toronto, August 2006

typically show transformations that re-orient or slightly shift these constituents. This suggests that the statistics of the set of unordered images should be kept in mind, when trying to model image transformations, and that the task of learning about transformations should be tied in with that of learning image statistics.

In order to learn about image transformations from training data, we construct a generative model that tries to predict the current (output) image in a stream of observations from the previous (input) one. The model contains a set of latent "mapping" units, that can develop efficient codes for the observed transformations, analogous to the hidden units in generative models of still images. In contrast to standard generative models however, the filters that the model learns for the current image are *conditioned* on the previous image in the stream. The model is thus trained to predict transformed versions of input images, which forces it to develop efficient encodings for the encountered transformations. At test time, the transformation can be inferred from a given pair of input-output images, as the conditional distribution over the latent mappings units.

To be able to capture all the potential dependencies between the transformation, input, and output units, the three types of unit form three-way cliques in the graphical model. As a result, the task of performing feature *extraction* is tied to the task of feature *mapping*, and both are learned simultaneously.

Once a model for transformations has been trained, there are many potential uses for it. A difficult ongoing problem in pattern recognition is dealing with invariances. We show how *learning* about transformations can greatly improve the performance in a recognition task in which the class labels are invariant under these transformations.

1.1 Related work

While there has not been much work on learning to encode transformations, the idea of using mapping units to encode transformations is not new and dates back at least to [1], which describes an architecture for modeling simple transformations of letters. However, no learning is performed in the model. A line of research that was inspired by this approach used mapping units to modulate feature extraction pathways (see [2]), but without learning.

Another early, biologically inspired, architecture for modeling transformations of images is described in [3]. The model was able to learn some simple synthetic transformations, but no applications were reported.

[4] and [5] constructed systems to model domain-specific transformations (the first for modeling motion, the latter for modeling geometric invariances). Both models were hand-crafted to work in their specific domains, and not trained to perform feature extraction, but they showed some interesting results on real-world tasks.

Our model is a type of higher-order Boltzmann machine [6] and can also be viewed as a conditional version of a restricted Boltzmann machine (RBM). It therefore bears some resemblances to [7], which used a kind of conditional RBM in a pixel labelling task. In that model, however, the dependence on the inputs is simply in the form of biases for the output units, whereas in our model, input, hidden and output units form three-way cliques, so the effect of an input unit is to modulate the *interaction* between transformation units and output units. As result, *filters* on the outputs, and not just

the outputs themselves, depend on the inputs, which is crucial for the task of learning image transformations.

2 Gated Boltzmann machines

The basic idea of our model is to predict the next observation in a stream of observations, and to use *hidden variables* to capture the many possible ways in which the next observation can depend on the previous one.

2.1 The model

To simplify the exposition we consider binary units for now, and show later how to deal with more general distributions. To model the probability of an output-image (or patch) $\boldsymbol{y}$, given an input image (or patch) $\boldsymbol{x}$, we consider an energy-function that combines all components of input and output images. To explicitly capture the many possible ways in which the outputs can depend on the input, we introduce an additional vector of binary hidden variables $\boldsymbol{h}$.

A simple energy function that captures all possible correlations between the components of $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{h}$ is

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_{ijk} W_{ijk} x_i y_j h_k, \tag{1}$$

where W_{ijk} are the components of a three-way parameter-"tensor" $\boldsymbol{W}$, that learns from training data to weight the importances of the possible correlations. The components x_i, y_j of $\boldsymbol{x}$ and $\boldsymbol{y}$ can be either pixel intensities or higher-level descriptors such as the outputs of non-linear filters. The negative energy $-E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x})$ captures the compatibility between the input, output and hidden units.

Note that, in the way that the energy is defined, each hidden unit h_k can 'blend in' a slice $W_{..k}$, of $\boldsymbol{W}$, which defines a linear mapping from inputs to outputs. Therefore, when we *fix* all hidden units, we obtain simply a linear mapping as transformation if the output units are linear. However, in practice we will derive a probability distribution over the set of hidden and output units, and then *marginalize* over all possible mappings as we describe below, which gives rise to highly non-linear and possibly (if we use more than one hidden unit) compositional mappings.

Using this energy function, we can now define the joint[1] distribution $p(\boldsymbol{h}, \boldsymbol{y}|\boldsymbol{x})$ over outputs and hidden variables by exponentiating and normalizing:

$$p(\boldsymbol{y}, \boldsymbol{h}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp(-E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x})) \tag{2}$$

where

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}, \boldsymbol{h}} \exp(-E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x})) \tag{3}$$

[1] We deliberately do not try to model the input, but rather condition on it – freeing us from many of the independence assumptions that a corresponding generative model would need to make to be tractable.

is a normalizing constant, that depends on the input image $\boldsymbol{x}$. To obtain the distribution over output images, given the input, we marginalize and get:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{h}} p(\boldsymbol{y}, \boldsymbol{h}|\boldsymbol{x}) \tag{4}$$

Note that in practice we cannot actually compute $p(\boldsymbol{y}|\boldsymbol{x})$ or $Z(\boldsymbol{x})$ exactly, since both contain sums over the exponentially large number of all possible hidden unit instantiations (and output unit instantiations, for $Z(\boldsymbol{x})$). In practice however, we do not actually need to compute any of these quantities to perform either inference or learning, as we shall show.

Inference at test time consists of inferring the transformation, or equivalently its encoding $\boldsymbol{h}$, from a *given* pair of observed images $\boldsymbol{x}$ and $\boldsymbol{y}$. But from Eqs. 1 and 2 it follows easily that

$$p(h_k = 1|\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \frac{1}{1 + \exp(-\sum_{ij} W_{ijk} x_i y_j)} \tag{5}$$

for every mapping unit h_k. This shows that the mapping units are *independent* binary variables given the input-output image pair, and can be computed efficiently. Similarly, for the distribution over outputs, when input and mapping units are given, we get:

$$p(y_j = 1|\boldsymbol{x}, \boldsymbol{h}) \quad = \quad \frac{1}{1 + \exp(-\sum_{ik} W_{ijk} x_i h_k)} \tag{6}$$

In practice, to be able to model affine and not just linear dependencies, it is useful to add biases to the output and hidden units. In contrast to unconditional models, where biases are simple additive components that can "shift" the activity level of a unit, in this more general model we can use also gated biases, that shift an activity *conditionally*. An energy function containing both gated biases, as well as standard biases, can take the form[2]:

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_{ijk} W_{ijk} x_i y_j h_k - \sum_{jk} W^{\boldsymbol{yh}}_{jk} y_j h_k - \sum_{j} W^{\boldsymbol{y}}_{j} y_j - \sum_{k} W^{\boldsymbol{h}}_{k} h_k \tag{7}$$

2.2 Learning

To train the probabilistic model, we maximize the average conditional log likelihood $L = \frac{1}{N} \sum_{\alpha} \log p(\boldsymbol{y}^{\boldsymbol{\alpha}}|\boldsymbol{x}^{\boldsymbol{\alpha}})$ for a set of N training pairs $(\boldsymbol{x}^{\boldsymbol{\alpha}}, \boldsymbol{y}^{\boldsymbol{\alpha}})$. In this paper we consider gradient based optimization for this purpose. The gradient of the log-likelihood for each training case is the difference of two expectations:

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_{\alpha} \left\langle \frac{\partial E(\boldsymbol{y}^{\boldsymbol{\alpha}}, \boldsymbol{h}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial \boldsymbol{W}} \right\rangle_{\boldsymbol{h}} - \left\langle \frac{\partial E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial \boldsymbol{W}} \right\rangle_{\boldsymbol{h}, \boldsymbol{y}} \tag{8}$$

[2]Even more general forms of bias are possible: Any set of weights that affects either one or two groups of units (inputs-to-hiddens, inputs-to-outputs, hiddens-to-outputs, just hiddens, or just outputs) can be considered a "generalized bias" in this model. In most of our experiments we have used all possible connections. But the energy function defined in Eq. 7 (containing only the single-node biases and the hidden-to-output biases) is usually sufficient to get good results.

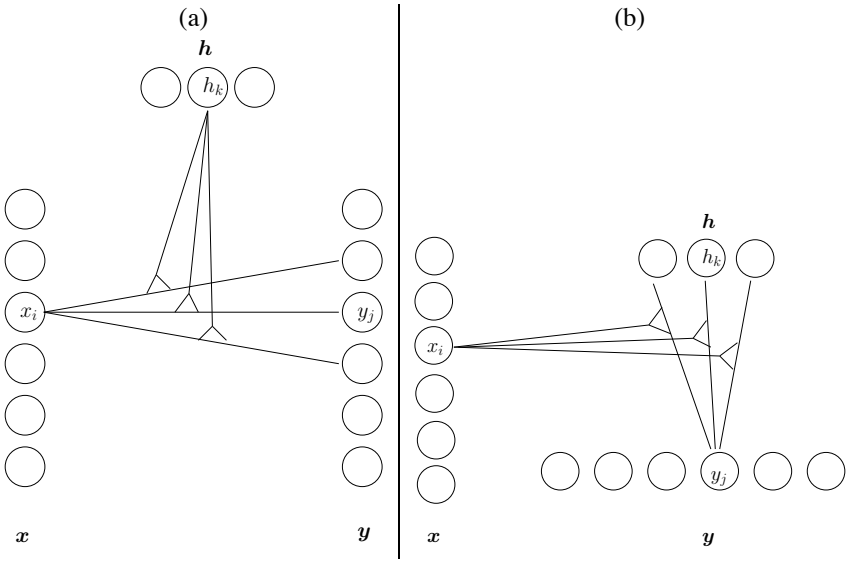

Figure 1: Two views of the basic model. (a) Gated regression: Hidden units can add slices $W_{..k}$ into a blend of linear transformations. (b) Modulated filters: Input units gate a set of basis functions that learn to reconstruct the output.

where we let $\boldsymbol{W}$ denote the set of all weights (including any biases, if present). The first expectation is over the posterior distribution over mapping units and can be computed efficiently using Eq. 5. The second is an expectation over all possible output/mapping instantiations and is intractable. Note however that, because of the conditional independences of $\boldsymbol{h}$ given $\boldsymbol{y}$, and $\boldsymbol{y}$ given $\boldsymbol{h}$ (see previous section), we can easily sample from the conditional distributions $p(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{y})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{h})$.

Gibbs sampling therefore suggests itself as a way to approximate the intractable term. Moreover, it can be shown that it is sufficient, and often advantageous, to perform only very few Gibbs iterations, if we start sampling at the training data-points themselves. This scheme of optimizing an undirected graphical model is known as contrastive divergence, and has been applied in several vision applications before (see [8], [9], [10], for example).

2.3 Two views

Since the model defines a conditional distribution over outputs, it can be thought of as an autoregressive model. In particular, Eq. 4 shows that it is a kind of mixture of experts, with a very large number of mixture components (exponential in the number of mapping units). Unlike a normal mixture model, the exponentially many mixture components share parameters which is what prevents it from overfitting. The number of parameters scales only linearly, not exponentially, with the number of mapping units.

Each binary mapping unit h_k that is active effectively 'blends' in a slice $W_{..k}$ of

the weight tensor $\boldsymbol{W}$ into the mixture (see Eq. 1). The model can therefore *compose* a given transformation from a set of simpler transformations, which is crucial for modeling many real-world transformations, in which different parts of an image may transform in different ways. Likewise at test time, the hidden units *de-compose* an observed transformation into its basic components by measuring correlations between input- and output-components. The importance that hidden unit h_k attributes to the correlatedness (or anti-correlatedness) of a particular pair x_i, y_j is determined by W_{ijk}, as can be seen also from Eq. 5: If W_{ijk} is positive then a positive correlation between x_i and y_j will tend to excite unit h_k, and a negative correlation tend to inhibit it. Importantly, the task of *defining* the set of basis transformations that is needed for some specific task at hand, is considered to be a domain-specific problem, and is therefore left to be solved by learning.

An alternative view of the model is shown in figure 1 (b): Each given, fixed input image $\boldsymbol{x}$ defines a bi-partite network (known as restricted Boltzmann machine, see [11], for example), in which input-dependent filters $\hat{W}_{jk} = \sum_i W_{ijk} x_i$ are used to capture spatial correlations in the image. The filters are input-weighted sums of slices $W_{i..}$ of the weight tensor $\boldsymbol{W}$. Folding the inputs into the weights this way lets us rewrite the energy in the form of a (case-dependent) RBM as:

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_{jk} \hat{W}_{jk} y_j h_k - \sum_j W_j^{\boldsymbol{y}} y_j - \sum_k W_k^{\boldsymbol{h}} h_k \tag{9}$$

(Note that, as before, we can also add generalized biases to the input-modulated weights, by defining the weights alternatively as: $\hat{W}_{jk} = \sum_i W_{ijk} x_i + W_{jk}^{\boldsymbol{hy}}$)

In other words, the model defines a bipartite conditional random field over output images and mappings. Instead of specifying spatial correlations ahead of time, as would be done for example with a Markov random field, here the possible correlations are learned from training data and can be domain-specific, input-dependent, and possibly long-range.

The fact that the model defines an RBM once the input has been fixed, is very convenient for inference and learning, since it allows us to make use of all the available machinery that has been developed for similar tasks in RBMs. Furthermore, approaches for extending RBMs also carry over easily to GBMs (see Section 2.5, for example).

2.4 Example: Transformations of binary images

As a proof of concept, we trained a model with 20 mapping units on synthetic videos showing random binary images of size 10×10 pixels that are shifted by one pixel in a random direction in each frame (see figure 2, leftmost two columns). To generate the data, we first generated a set of initial images by turning on each pixel in an image randomly with probability 0.1. Then, repeatedly we chose for each image a direction randomly from the set {up, down, left, right, up-left, up-right, down-left, down-right, no shift}, and shifted the images, filling in newly appearing edges randomly, and independently as before with probability 0.1 We trained the model on batches containing 100 image pairs each. Since the image pairs were generated on the fly, training was

online, with a gradient update after each presentation of a batch[3]. We trained the model on several thousand batches, which takes a few minutes on a Pentium 4 with 3.4 GHz.

After training, the model can infer 'optical flow' from a pair of test-images by computing the hidden unit activations using Eq. 5. The model represents the flow that it observes implicitly in these mapping unit activations. But it is possible to visualize an approximation of the models 'percept', by drawing for each input-pixel an arrow to the output-pixel to which the input-pixel connects the most according to the (marginalized) weight-tensor W.

Figure 2 shows seven example image pairs and the inferred 'max-flow'-fields. The model is able to consisently infer the correct motion over the whole patch (except at edges that move into the image, where pixels are random and cannot be predicted).

Given a set of hidden unit activations $\boldsymbol{h}$, we can apply the encoded transformation to a new, previously unseen image $\boldsymbol{x}^{\text{new}}$ by computing $p(\boldsymbol{y}|\boldsymbol{h}, \boldsymbol{x}^{\text{new}})$. The right-most column in the figure depicts these transfered transformations using gray values to represent the probability of turning a pixel 'on'.

In this example, a mixture of experts, *ie*. a model that contains a single, multinomial mapping unit, would in principle be sufficient, because the set of possible transformations has a small size (9 in this case). Figure 3 shows a variation of the experiment, where the transformations are *factorial* instead. The image pairs are generated by transforming the top and bottom halves of the input images independently of each other. We used 5 possible transformations for this experiment (shift left, right, up, down and no shift), and 10 mapping units as before. While 10 mixture components in an ordinary mixture model would not be sufficient to model the resulting $5 \times 5 = 25$ transformations, the GBM finds a factorial code that can model the transformations, as can be seen from the figure. While in these toy-examples there are no correlations between the pixels in a single image, in natural images we would expect such correlations to provide further cues about the transformations. We revisit this kind of experiment using natural images in Section 4.1.

[3]Since training data can be generated on the fly, the amount of available training data is essentially unlimited. We would like to point out that training with this data set would be rather difficult with non-parametric methods, such as kernel methods.

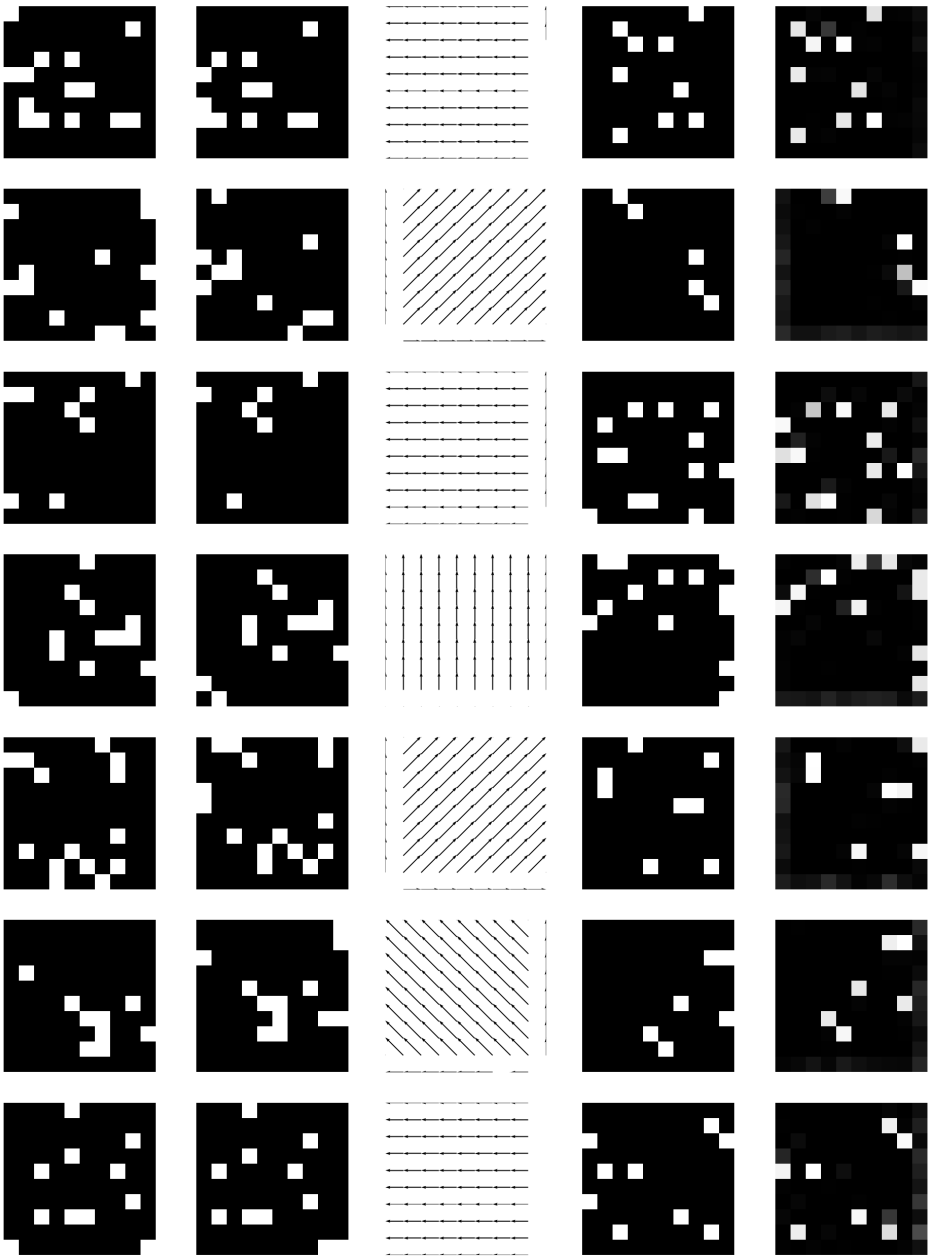

Figure 2: Columns (left to right): Input images; output images; inferred flowfields; random target images; inferred transformation applied to target images. For the transformations (last column) gray values represent the probability that a pixel is 'on' according to the model, ranging from black for 0 to white for 1.

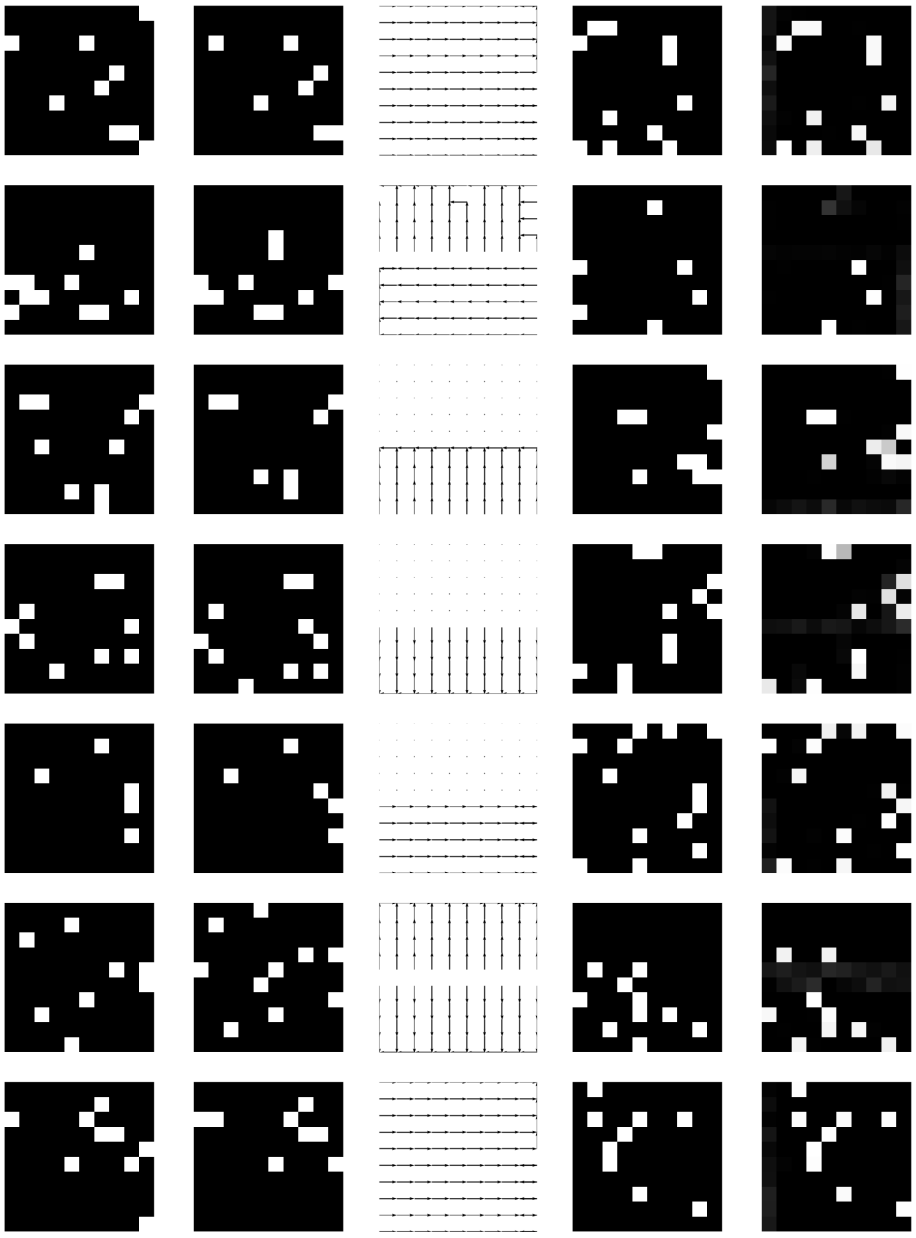

Figure 3: Factorial transformations. Display as in figure 2.

2.5 Generalization to non-binary units

The model defined so far defines a binary distribution over output and mapping units. However, binary values are not always the best choice, and it is often desirable to be able to use more general distributions, for either output or mapping units. Especially continuous values for the outputs can be useful in image modelling tasks, but also other choices (for either outputs or hidden units) are imaginable, such as multinomial, Poisson, or others.

Modifying the model in order to deal with continuous outputs, while keeping the hidden units binary, for example, can be achieved straightforwardly, following the same approach that is used for continuous standard RBMs [11]: We re-define the energy as (similarly when using the other kinds of generalized bias):

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = \frac{1}{2\nu^2} \sum_j (y_j - W_j^{\boldsymbol{y}})^2 - \frac{1}{\nu} \sum_{ijk} W_{ijk} x_i y_j h_k - \sum_k W_k^{\boldsymbol{h}} h_k, \tag{10}$$

and then define the joint distribution as before (Eq. 2). Now, while the conditional distribution over hidden units $p(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{y})$ remains the same (because the additional term cancels out after exponentiating and conditioning), it is straightforward to show that the distribution over outputs turns into a Gaussian (see [11] for details):

$$p(y_j|\boldsymbol{x}, \boldsymbol{h}) = \mathcal{N}\left(y_j; \nu \sum_{ik} x_i h_k W_{ijk} + W_j^{\boldsymbol{y}}, \nu^2\right) \tag{11}$$

We use a spherical Gaussian for the outputs here, but we could also use a different variance ν_j for each output-component.

Since the conditional distribution is a Gaussian, that is independent across components y_j, Gibbs sampling is still straightforward in the model. Note, that the marginal distribution is *not* Gaussian, but a mixture of exponentially many Gaussians. As before, it is intractable to evaluate the marginal, so it is fortunate that it does not need to be evaluated for either learning or inference. Note also, that the inputs $\boldsymbol{x}$ are always conditioned on in the model. They therefore do not need to be treated differently than for the binary case. Any scaling properties of the inputs can be absorbed into the weights, and are therefore taken care of automatically during training though learning is faster and more stable if the scales are sensible.

Using Gaussian outputs allows us to perform *regression* tasks, in which the outputs are high-dimensional and structured. It therefore constitutes an interesting extension to recent work on classification with structured outputs (see [12]). In contrast to the recent kernel based attempts in this direction (*eg*. [13]) our model can be trained on much larger datasets, and is capable of online learning.

While extending the model to deal with Gaussian outputs is useful for many problems, we do not need to stop there. In fact, it is easy to extend the model such that it uses any member of the exponential family as the conditionals $p(\boldsymbol{y}|\boldsymbol{h}, \boldsymbol{x})$ or $p(\boldsymbol{h}|\boldsymbol{y}, \boldsymbol{x})$. [9] describe a framework for constructing RBMs with arbitrary exponential family distributions as the conditionals, and their approach can be applied similarly to GBMs, because once we condition on an input the model simply takes the form of an RBM ('View 2' in Section 2.3).

Until now we have represented each state of a unit using a single binary value. The reason is that the sufficient statistics of binary variables are their binary states themselves. The sufficient statistics of more general random variables can be more complicated and can contain more than just one value. (Consider a Gaussian, for example, whose sufficient statistics are given by its mean and variance.)

We therefore introduce indices a and b and represent the variables y_j and h_k using their sufficient statistics $f_{ja}(y_j)$ and $g_{kb}(h_k)$, respectively. The joint distribution under this representation can then be re-defined as:

$$p(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) \propto \exp\left(-\sum_{jakb} \hat{W}_{ja}^{kb} f_{ja}(y_j) g_{kb}(h_k) - \sum_{ja} W_{ja}^{\boldsymbol{y}} f_{ja}(y_j) - \sum_{kb} W_{kb}^{\boldsymbol{h}} g_{kb}(h_k)\right), \tag{12}$$

where $\hat{W}_{ja}^{kb}$ are the modulated weights that we obtain similarly as before, by 'folding' the inputs into case-independent weights: $\hat{W}_{ja}^{kh} = \sum_i W_{ijakb} x_i$.

Eq. 12 is again simply a standard (but case-dependent) RBM. As before, and as in a standard RBM, the conditional distributions under this joint distribution decouple into products of independent distributions, that now are exponential family distributions themselves, and whose parameters are just 'shifted' versions of the biases:

$$p(y_j|\boldsymbol{h}; \boldsymbol{x}) \quad \propto \quad \exp\left(\sum_a \left[\sum_{kb} \hat{W}_{ja}^{kb} g_{kb}(h_k) + W_{ja}^{\boldsymbol{y}}\right] f_{ja}(y_j)\right) \tag{13}$$

$$p(h_k|\boldsymbol{y}; \boldsymbol{x}) \quad \propto \quad \exp\left(\sum_b \left[\sum_{ja} \hat{W}_{ja}^{kb} f_{ja}(y_j) + W_{kb}^{\boldsymbol{h}}\right] g_{kb}(h_k)\right) \tag{14}$$

Note that we let the inputs modulate the couplings as before, instead of modulating the sufficient statistics. Fixing the sufficient statistics has the advantage that it keeps learning and inference simple. Since the conditionals decouple as previously for the binary case, Gibbs sampling is still straightforward. In the practical implementation, basically all that changes for training the model is the routines for sampling from the conditionals.

3 Extensions

3.1 Fields of Gaussian Experts

Until now we have considered a single global model, that connects each input pixel (or feature) with each output component and we have used *patches* to train the model on real-world data.

For images that are much larger, connecting all components is not going to be tractable. The most simple solution for natural video data is to restrict the connections to be local, since the transformations on these data-sets are typically not very long-range. In many real-world tasks (albeit not in all) it is reasonable to assume that the kind of transformations that occur in one part of the image could occur in principle also

in any other. Besides restricting connections to be local, it can therefore makes sense to also use some kind of weight-sharing, so that we apply essentially the same *model* all over the image, though not necessarily the same particular transformation all over the image.

We therefore define a single patch-model, as before, but define the distribution over the whole output-images as the *product* of distributions over patches centered at each output-pixel. Each patch (centered at some output-pixel y_j) contains its own set of hidden units h_k^j. Formally, we simply re-define the energy to be (we drop the biases here for simplicity)

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_s \sum_{ijk} W_{ijk} x_i^s y_j^s h_k^s, \tag{15}$$

where s ranges over all sites, and x_i^s denotes the (i^{th}) component of $\boldsymbol{x}$ in site s (analogously for $\boldsymbol{y}$ and $\boldsymbol{h}$). Inferring the hidden unit probabilities at some site s, given the data, can be performed exactly the same way as before independently of the other sites, using Eq. 5. When inferring the data distribution at some site s, given the hiddens, some care needs to be taken because of overlapping output patches. Learning can be performed the same way as before using contrastive divergence.

This approach is the direct conditional analogue to modeling a whole image using patch-wise RBMs as used in [10], [14] for the non-conditional case. A similar approach for simple conditional models has also been used by [7].

3.2 Neural network premappings

The fact that the model defines a conditional (and not joint) distribution over outputs makes it easy to develop extensions, where the model learns to pre-process the inputs prior to transforming them. We can define feature functions $\phi_i(\boldsymbol{x})$, and re-define the energy (Eq. 1) to $E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_{ijk} W_{ijk} \phi_i(\boldsymbol{x}) y_j h_k$ (similarly when using biases). Training the whole model then consists in adapting both the mapping parameters $\boldsymbol{W}$, and the parameters of the feature functions simultaneously. If we use differentiable functions as features, we can use back-propagation to compute the gradients ([15],[16]).

Pre-processing the input in this way can have several advantages. One practical advantage is that we can use dimensionality reduction to alleviate the quadratic computational complexity of the model. Another is that good feature extraction can improve generalization accuracy. In contrast to using the responses of fixed basis functions, such as PCA features, training the whole architecture at once allows us to extract 'mappable features' that are optimized for the subsequent transformation with the GBM.

4 Experiments

4.1 The transformation-fields of natural videos

Recall (Eq. 5 for binary units, Eq. 14 for more general units) that mapping unit probabilities are inferred by measuring correlations between input- and output-components.

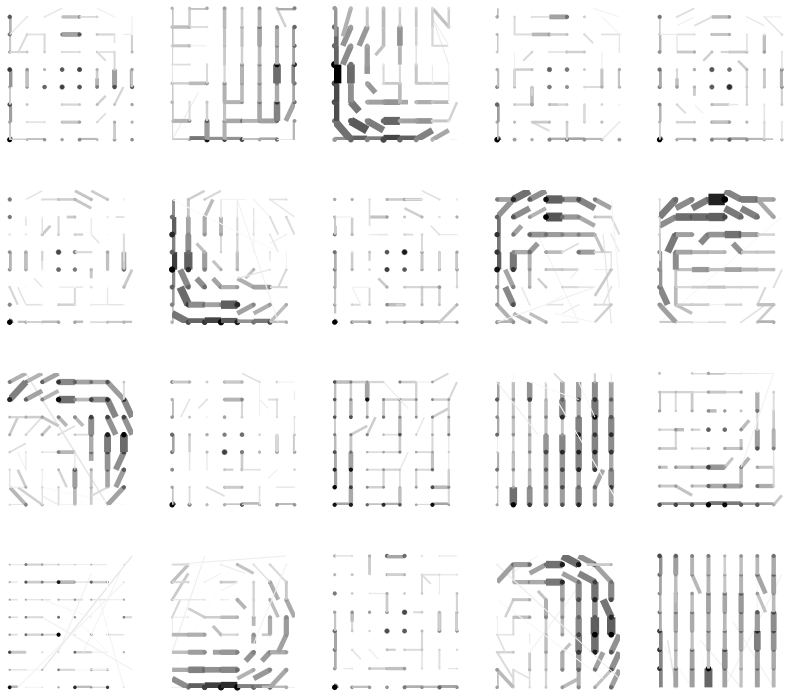

Figure 4: 20 basis flowfields learned from synthetic image transformations. The flowfields show, for each input pixel, the strengths of its positive outgoing connections, using lines whose thicknesses are proportional to the strengths.

The mapping units act like Reichhardt detectors [17]. In contrast to simple standard Reichhardt detectors the mapping units make use of *spatial pooling*: Each detector (mapping unit) connects a set of input units with a set of output units, and can therefore generalize over larger image regions and deal with the presence of noise. The particular correlation patterns between input pixels and output pixels that are prevalent in a certain domain are encoded in the weight tensor $\boldsymbol{W}$. Most importantly, these patterns, and thereby also the distribution of responsibilities across mapping units, are not assumed to be known, but are *learned* from training data.

To see which forms these 'Reichhardt pools' take on when trained on real world data, we used a database of digitized television broadcasts[18]. The original database contains monochrome videos with a frame-size of 128×128 pixels, and a frame rate of 25 frames per second. We reduced the frame-rate by a factor of 2 in our experiments, *ie*. we used only every other frame. Further details about the database can be found in [18].

Learning synthetic transformations: First, to see whether the model is able to discover very simple transformations, we trained it on synthetically generated transformations of the images. We took random-patches from the video-database described

above (without considering temporal information) and generated sequences by transforming the images with shifts and rotations. We used 20 mapping units and trained on images of size 8×8 pixels. We used the pixel intensities themselves (no feature extraction) for training, but smoothed the images with a Gaussian filter prior to learning.

Figure 4 displays resulting 'excitatory basis-flow-fields' for several mapping units, by showing for each mapping unit h_k the strength of the *positive* connections W_{ijk}, between pixel i and pixel j, using a line whose thickness is proportional to the connection strength. We obtained a similar plot (but poled in the opposite direction) for negative connections, which means that the model learns to locally shift edges. (See below for further results on this.)

Furthermore, the figure shows that the model infers locality, *ie*. input pixels are connected mostly to nearby pixels. The model decomposes the observed transformation across several mapping units. (Note that this display loses information, and is used mainly to illustrate the resulting flow-fields.)

Broadcast videos: To train the model on the actual videos we cut out pairs of patches of size 20×20 pixels at the same random positions from adjacent frames. We then trained the model to predict the second patch in each pair from the first, as described previously. To speed up training, we used 100 PCA-coefficients instead of the original pixel-intensities as the patch-representations $\boldsymbol{x}$ and $\boldsymbol{y}$, reducing the input-dimensionality from 400 to 100. We used no other forms of preprocessing in these experiments. (It is possible to obtain similar results with the original, pixel-wise, image representations instead of PCA-coefficients, in which case Gaussian smoothing and/or whitening can help the learning.)

A way of visualizing the learned basis flowfields is shown in figure 5. The figure shows that the model has developed sets of local, conditional edge-filters. That is, input pixels only have significant weights to output pixels that are nearby and the weights form an oriented edge filter. The fact that the model decides to latch onto edges to infer information about the observed transformation is not surprising, given that image gradient information is essential for inferring information about optical flow. Note however, that this information is learned from the database and not handcoded. No sparsity constraints were required to obtain these results. The database – being based on broadcast television – shows many small motions and camera-shifts, and contains motion as a predominant mode of variability.

More interesting than the learned motion edge-features is the fact that adjacent input-pixels tend to be mapped similarly by given mapping units, which shows that the model has learned to represent mostly global motion within the 20×20-patch, and to use spatial pooling to infer information about the observed transformation.

Given the learned model, it is straightforward to generate dense flow-fields as in Section 2.4. The plots in the top row of figure 6 show a pair of two adjacent example time-frames cropped randomly from the video-database, and showing a more or less even right-shift over the whole patch. To generate the flow-field, as before, at each input-pixel position in the center region of the image (we left out a frame of 4 pixels width, where the field cannot be inferred precisely) an arrow is drawn that shows to which output-pixel it connects the most[4] (bottom row of the figure). The resulting flow-

[4]Note that the restriction to integer flow here is not a restriction of the model per se – in particular, since

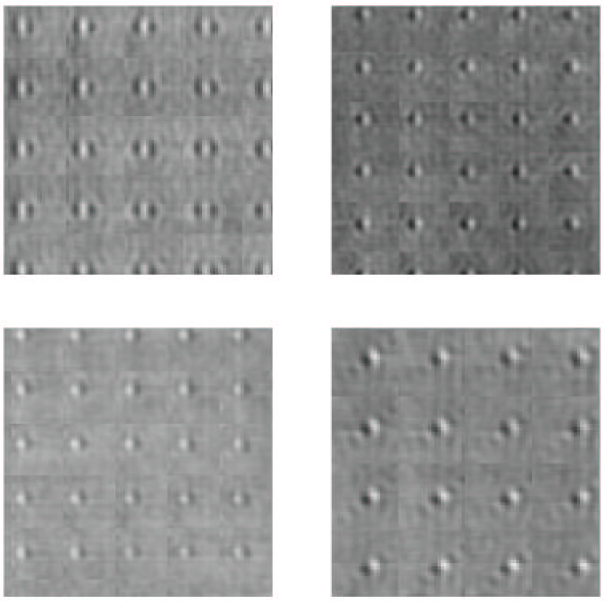

Figure 5: A visualization of parts of the learned basis flowfields W_{ijk} for four different hidden units, h_k. For each hidden unit, the input pixels in a small patch are laid out in a coarse grid. At each grid location, there is an intensity image that depicts the weights from that input pixel to all of the output pixels. The intensities range from black for strong negative weights to white for strong positive ones. We invert the PCA encoding before showing the results.

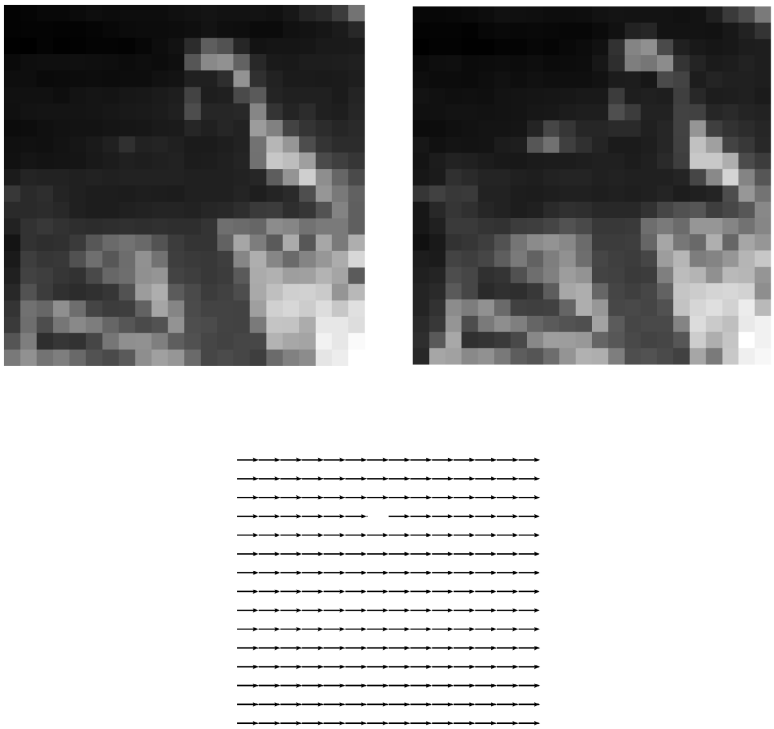

Figure 6: Typical image patch pair from the video database, and the inferred flow field. Spatial pooling facilitates generalization and noise suppression.

field shows that the model infers that there was a more or less global motion within the patch, even though corresponding pixel intensities vary considerably between frames, and there are large homogeneous regions. The reason that the model infers a global motion is that it considers it to be the most probable, given the observed evidence, and that such motions are typical in the dataset, whose log-probability is what is being optimized during training.

Note that we do not necessarily advocate using this method to infer dense flow-fields such as the one shown. Rather, the flow-information is represented implicitly here in the form of hidden unit probabilities, and condensing it to the max-flow field would usually mean that potentially useful information gets lost. Also, the kinds of transformations that the model learns are not restricted to motion (as we show below), and are therefore not necessarily local as in the example. To make use of the implicitly encoded information one can use *learning* on the hidden units instead (As done in [4], for example).

the described model was trained on basis-transformed patches. The flow-field is used mainly for illustration purposes as described in the main text. The full flow representation is given by the mapping unit activations themselves.

4.2 Learning an invariant metric

Once a model has been trained to recognize and encode image transformations, it is possible to perform recognition tasks that are *invariant* under those transformations, by defining a corresponding invariant metric *w.r.t*. the model. Since the GBM model is trained entirely from data, there is no need to provide any knowledge about the possible transformations to define a metric, (in contrast to many previous approaches, such as [19]).

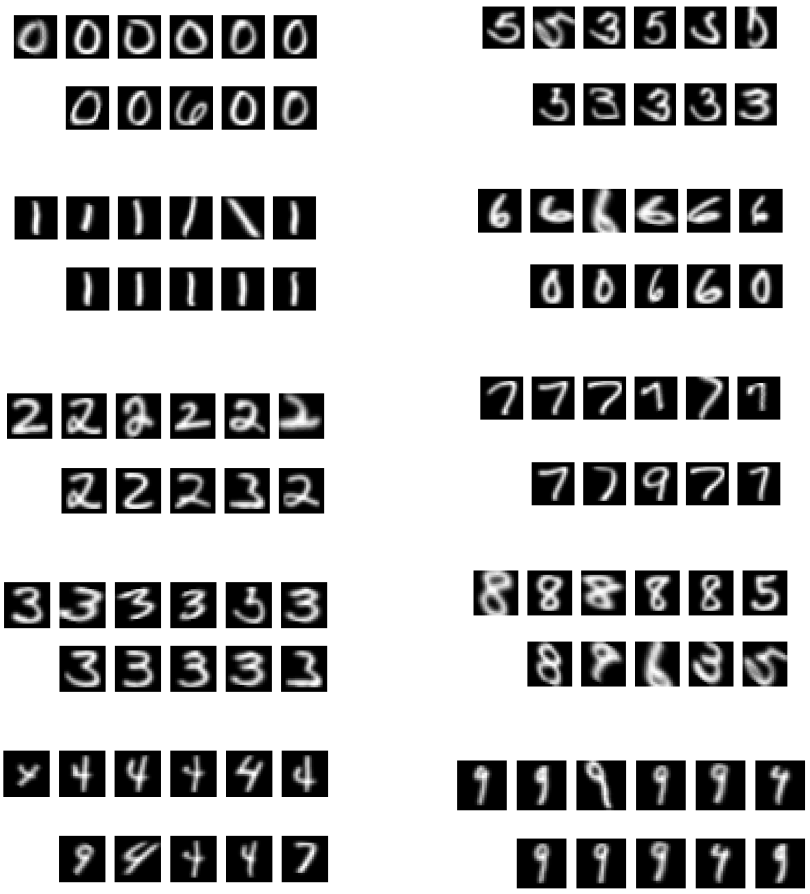

Figure 7: The five nearest neighbors of a digit using two different metrics. The left-most digits in each row of six are query-cases. The other five digits in that row are the five nearest neighbors according to the learned model. Below these are the five nearest neighbors according to Euclidean distance in pixel space.

An obvious way of computing the distance between two images, given a trained GBM model, is by measuring how well the model can transform one image into the

other. If it does a good job at modelling the transformations that occur in the distribution of image pairs that the data was drawn from, then we expect the resulting metric to be a good one.

A very simple, but as it turns out, very effective, way of measuring how well the model can transform an input image $\boldsymbol{x}$ into another image $\boldsymbol{y}$, is by first inferring the transformation, then applying it, and finally using Euclidean distance to determine how well the model did. Formally this amounts to using the following three-step algorithm:

1. Set $\hat{\boldsymbol{h}} = \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{x}, \boldsymbol{y})$
2. Set $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}, \hat{\boldsymbol{h}})$
3. Define $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|$,

where $d(\cdot, \cdot)$ is the resulting distance measure, which is strictly speaking not a distance since it is not symmetric. (It could easily be turned into a proper distance by adding the two opposite non-symmetric versions, but in many applications symmetry is not actually needed.) Note that both operations can be performed efficiently because of the independence of the conditional distributions.

It is interesting to note that we can interpret the procedure also as measuring how well the model can reconstruct an output $\boldsymbol{y}$ under the conditional distribution defined by the clamped input $\boldsymbol{x}$, using a one-step reconstruction similar to the one used during contrastive divergence learning. Points that lie in low-density (and correspondingly high-energy) regions tend to get 'pulled' towards higher density regions more strongly than points that already reside in high density regions, and therefore experience a larger shift in terms of Euclidean distance.

Figure 7 shows the nearest neighbors for a few test-cases from a digit dataset (see next section for a detailed description of the dataset) among three hundred randomly chosen training cases according to a GBM trained to predict affine transformations of the digits. We used a model with 30 mapping units and with a linear one-layer neural network premapping (see Section 3.2) to reduce the input dimensionality to 30 in order to speed up training. Both the model parameters and parameters of the pre-mapping were trained simultaneously. The premapping was trained by backpropagating the derivatives produced by contrastive divergence. The derivatives that are used to learn the bias of a unit can always be used to learn how this bias should depend on some other, given input. After training, we picked query-cases $\boldsymbol{y}$ from a test-set (that was not seen during training) and determined the five nearest neighbors among 300 randomaly chosen training cases using (i) the metric induced by the model and using (ii) Euclidean distance. The figure shows that, in contrast to Euclidean distance, the distance induced by the model cares little about pixel overlap and considers digits to be similar, if they can be transformed into each other.

4.2.1 Application to digit classification

In the task of digit classification, viewpoint invariance is usually not a central concern because typical datasets are composed of normalized images. This allows nearest neighbor classification in pixel space to obtain reasonably good results.

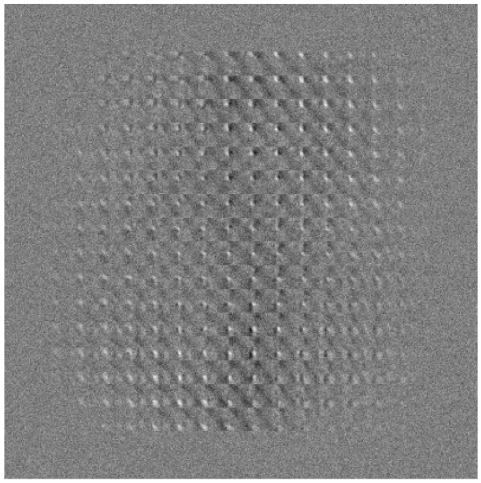

Figure 8: A transformation field learned from affine digits. Connection strengths are represented as in figure 5.

Here we modify the digit classification task by taking 5000 randomly chosen examples from the USPS-dataset (500 from each class) and generating 15000 extra examples using random affine transformations (3 for each case).

Predicting transformed digits from their originals requires quite different transformation fields than those required for predicting video images. Figure 8 shows a typical transformation field after training a GBM with 50 mapping units on the raw pixel images. One obvious feature is that the model is indifferent with regard to the edges of the image (where it encounters much less variability than in the middle).

We now consider the problem of predicting 10000 of the transformed digits from the 5000 original training points (for which we have labels available). The remaining 5000 transformations are (i) left aside to learn transformations in one set of experiments, (ii) included in the training-set for classification in another set of experiments. The first problem is one of transfer learning: We are given labels for the unmodified training cases, but the actual task is to classify cases from a different, but *related*, test set. The relation between the sets is provided only implicitly, in the form of correspondences between digits and their transformed versions.

We trained a GBM with 50 mapping units on 100 PCA-features on the transformations, by predicting the transformed versions of digits from the originals and vice versa. Figure 9 compares the nearest neighbor error rates on the 10000 test cases, obtained using either the Euclidean distance or the distance computed using the model of transformations. To compute nearest neighbors using the non-symmetric distance measure, we let the training cases be the inputs $\boldsymbol{x}$ in the three-step algorithm (previous section). In other words, we measure how well the model can transform prototypes (training cases) to the query cases.

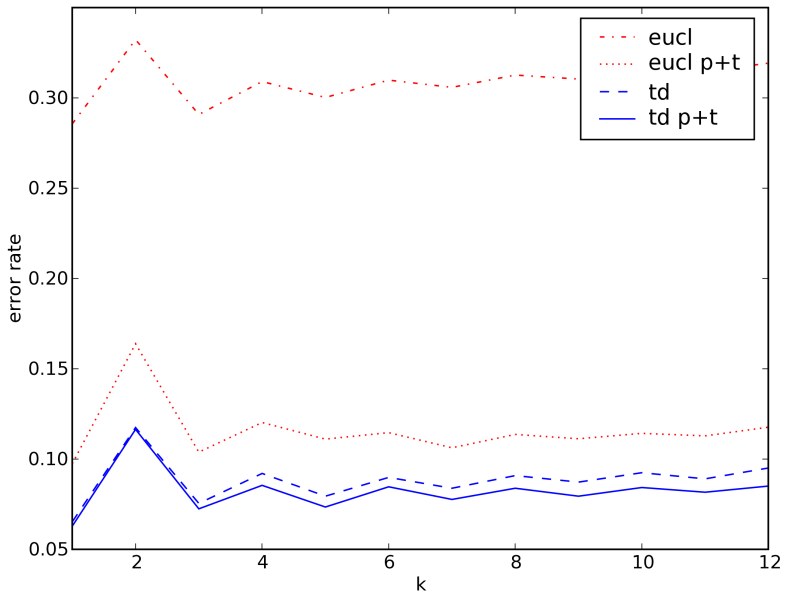

Figure 9: Error rates using k-nearest neighbors. The model induced distance is compared with the Euclidean distance for two different training sets. Adding affine transformed digits to the training set helps the Euclidean distance a lot, but it is of very little help to the learned distance because the model can learn about transformations from the basic training set.

In task (i) ('eucl' vs. 'td' in the plot) Euclidean distance[5] fails miserably, when transformed digits have to be predicted from the original dataset, while the transformation metric does quite well. In task (ii) ('eucl p+t' vs. 'td p+t' in the plot) where transformed versions are included in the training set, Euclidean distance improves significantly, but still does considerably worse than the transformation metric. The transformation metric itself gains only little from including the transformations. The reason is, that most of the available information resides in the way that the digits can be transformed, and has therefore already been captured by the model. Including transformed digits in the training set does not, therefore, provide a significant advantage and leaving them out does not entail a significant disadvantage.

[5]For the Euclidean distance results, we also compared with Euclidean distance in the 100-dimensional PCA-coefficient space and to normalized data, but Euclidean distance in the original data-space gives consistently the best results.

4.3 Image transformations

Once we have a way of encoding transformations, we can also apply the these transformation to previously unseen images. If we obtain the transformation from some 'source' image pair and apply it to a target image, we are basically performing an analogy. [20] discuss a model that performs these general kinds of 'image analogies' using a regression-type architecture. An interesting question is, whether it is possible to perform similar kinds of transformations using a general purpose generative model of transformations.

We used a source image-pair as shown in the top row in figure 12, generated by taking a publicly available image from the database described in [21], and applying an "artistic" filter that produces a canvas-like effect on the image. The image sizes are 512×512 pixels. We trained a field of gated experts model with 10 hidden units on the raw images. The target image is another image from the same database and is shown in the row below, along with the transformation, which we obtained by performing a few hundred Gibbs iterations on the output (the one-step reconstruction works, too, but more iterations improve the results a little). The blow-up shows that the model has learned to apply a somewhat regular, canvas-like structure to the output-image, as observed in the source image.

5 Conclusions

There has been surprisingly little work on the problem of learning explicit encodings of data transformations, and one aim of this paper is to draw attention to this approach which, we believe, has many potential applications. There is some resemblance between our approach and the bilinear model of "style" and "content" proposed in [22], but the learning methods are quite different and our approach allows the transformations to be highly non-linear functions of the data.

There are several interesting directions for future work. One is to learn mappings of multiple different types of feature simultaneously. Doing this at multiple scales could be interesting, especially for motion-analysis. Potential applications (other than dealing with invariances for discrimination) are video compression and denoising using the temporal structure. Another interesting problem is the construction of layered architectures in which mapping units are shared between two adjacent layers. This allows transformations that are inferred from pixel intensities to be used to guide feature-extraction, because features of an image that are easy to predict from features of the previous image will be preferred. While we have considered image transformations in this paper, the model can easily be trained on more general data types than images, and we are currently working on applications in other areas.

RBMs are closely related to autoencoder networks [11], and can be used to discover the low-dimensional manifolds along which real world data is often distributed. GBMs in turn learn *conditional* embeddings, by appropriately modulating the weights of an RBM with input data. (Note that, while we have restricted our attention to the probabilistic RBMs, we could similarly directly gate the weights of an autoencoder instead.) These 'morphable manifolds' provide a convenient way of including prior

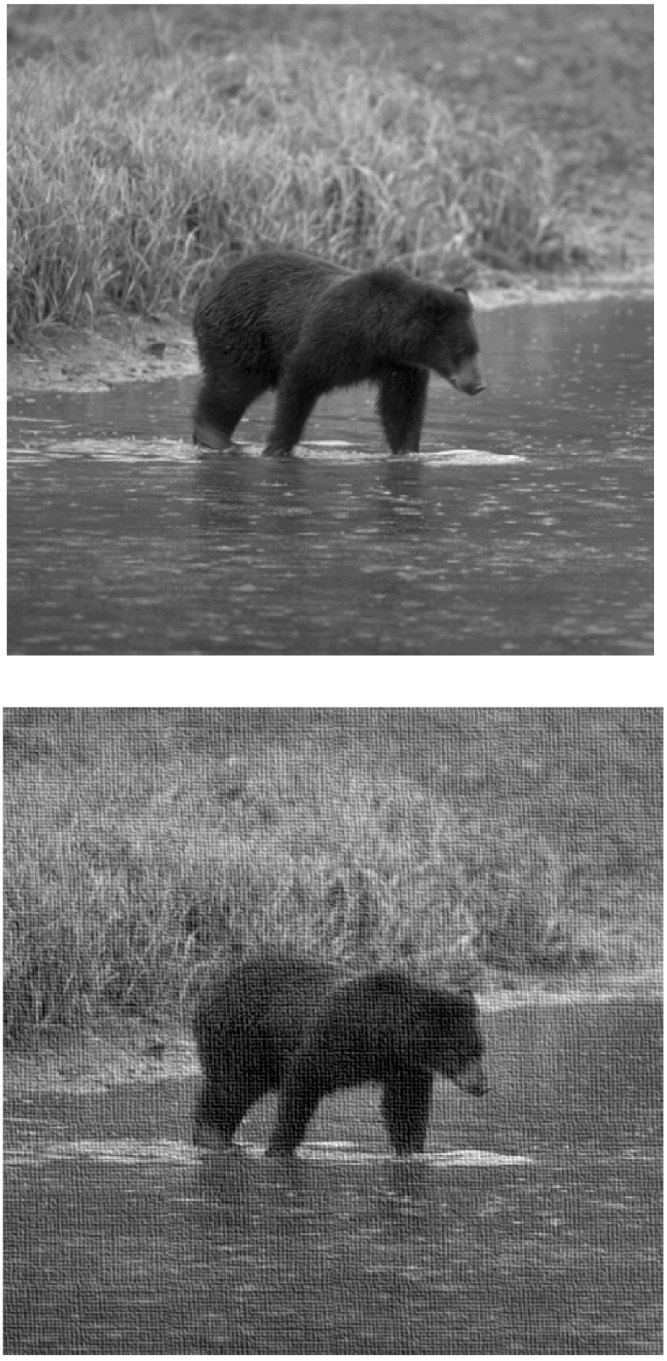

Figure 10: Image analogy (source image pair). Source image (top) and its transformation showing a canvas-like effect (bottom).

Figure 11: Image analogy (target image pair). Target image (top) and the learned transformation applied to the target image.

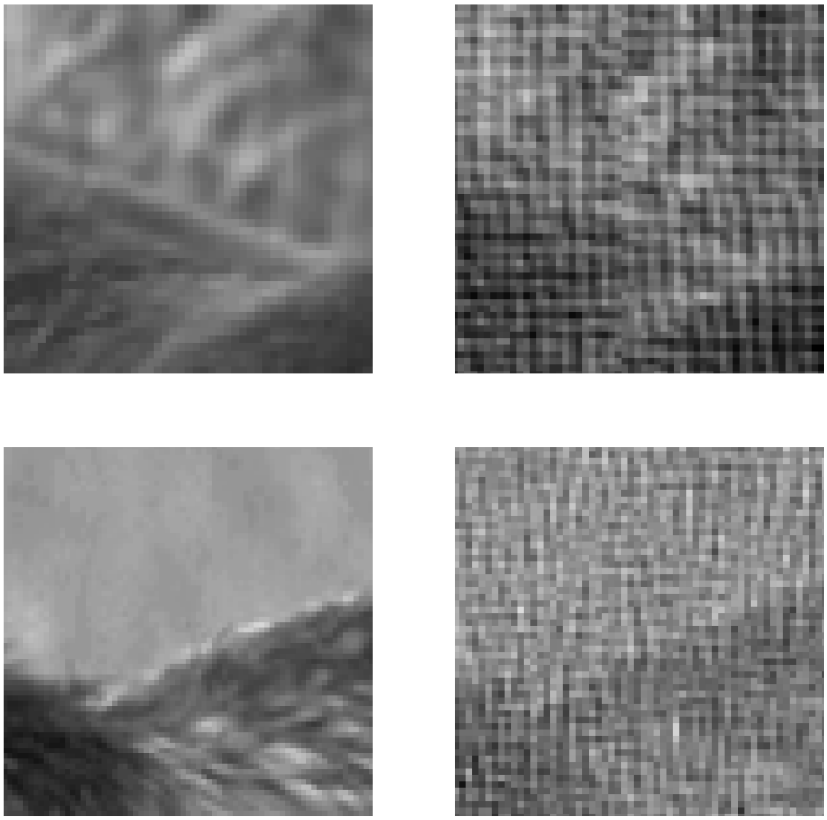

Figure 12: Image analogy blow-ups. Top row: Small patch from the source image and its transformation. Bottom row: Small patch from the target image and its transformation.

knowledge in embedding tasks, and can be useful in data analysis and visualization tasks. In contrast to previous, non-parametric approaches in this direction, such as [23] and [24], GBMs can be trained on much larger datasets and easily generalize the learned embeddings beyond the training data-points, which is what makes them so useful for discrimination tasks.

References

[1] Geoffrey Hinton. A parallel computation that assigns canonical object-based frames of reference. In *Proc. of the 7th IJCAI*, Vancouver, BC, Canada, 1981.

[2] Bruno Olshausen. *Neural Routing Circuits for Forming Invariant Representations of Visual Objects*. PhD thesis, Computation and Neural Systems, California Institute of Technology, 1994.

[3] Rajesh P.N. Rao and Dana H. Ballard. Efficient encoding of natural time varying images produces oriented space-time receptive fields. Technical report, Rochester, NY, USA, 1997.

[4] David J. Fleet, Michael J. Black, Yaser Yacoob, and Allan D. Jepson. Design and use of linear models for image motion analysis. *Int. J. Comput. Vision*, 36(3):171–193, 2000.

[5] Serge J. Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. Technical report, Berkeley, CA, USA, 2002.

[6] T.J. Sejnowski. Higher-order boltzmann machines. In J. Denker, editor, *Neural Networks for Computing*, pages 398–403. American Institute of Physics, 1986.

[7] Xuming He, Richard S. Zemel, and Miguel A. Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *CVPR 2004*, pages 695–702, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[8] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[9] Max Welling, Michal Rosen-Zvi, and Geoffrey Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 17*, pages 1481–1488. MIT Press, Cambridge, MA, 2005.

[10] Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. In *CVPR 2005*, pages 860–867, Washington, DC, USA, 2005. IEEE Computer Society.

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

[12] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[13] Y. W. Teh, M. Seeger, and M. I. Jordan. Semiparametric latent factor models. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 10, 2005.

[14] Stefan Roth and Michael J. Black. On the spatial statistics of optical flow. In *ICCV '05*, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, November 1998.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, October 1986.

[17] Werner Reichardt. Movement perception in insects. In Werner Reichardt, editor, *Processing of optical data by organisms and by machines*. Academic Press, New York, 1969.

[18] 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. *Proceedings: Biological Sciences*, 265(1412):2315–2320, 1998.

[19] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR '05*, pages 539–546, Washington, DC, USA, 2005. IEEE Computer Society.

[20] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *SIGGRAPH '01*, pages 327–340, New York, NY, USA, 2001. ACM Press.

[21] C. Grigorescu, N. Petkov, and M. A. Westenberg. Contour detection based on nonclassical receptive field inhibition. *IEEE Transactions on Image Processing*, 12(7):729–739, 2003.

[22] J.B. Tenenbaum and W.T. Freeman. Separating Style and Content with Bilinear Models. *Neural Computation*, 12(6):1247–1283, 2000.

[23] Roland Memisevic and Geoffrey Hinton. Multiple relational embedding. In *Advances in Neural Information Processing Systems 17*, pages 913–920. MIT Press, Cambridge, MA, 2005.

[24] Roland Memisevic. Kernel information embeddings. In *ICML 2006*, pages 633–640, New York, NY, USA, 2006. ACM Press.