Unsupervised Learning of Image Transformations

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

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

Abstract

We describe a probabilistic model for learning rich, distributed representations of image transformations. The basic model is defined as 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 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 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 [8], 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 [13]), but without learning.

Another early, biologically inspired, architecture for modeling transformations of images is described in [14]. The model was able to learn some simple synthetic transformations, but no applications were reported.

[3] and [1] 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 [17] and can also be viewed as a conditional version of a restricted Boltzmann machine (RBM). It therefore bears some resemblances to [5], 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, that take on values 0 or 1. We 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 $W_{\cdot\cdot k}$ 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 units, and then *marginalize* over all possible mappings as we describe below, which gives rise to highly nonlinear 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

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

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 guessing 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

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

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:

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

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. However, to simplify the notation we drop these,

[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 fully generative model would need to make to be tractable.

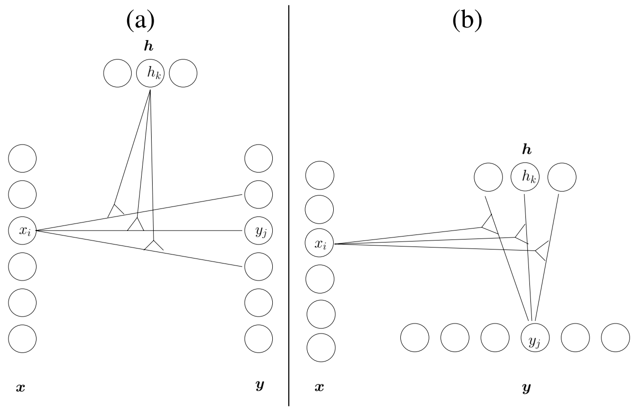

Figure 1. Two views of the basic model. (a) Gated regression: Hidden units can add slices $W_{\cdot\cdot k}$ into a blend of linear transformations. (b) Modulated filters: Input units gate a set of basis functions that learn to reconstruct the output.

and equivalently think of all the hidden and output vectors as having an additional input that always has an activity of 1 so that the bias is the weight on this input.

Eq. 5 shows that hidden unit probabilities are inferred using correlations between input- and output components, which shows that at test time each hidden unit effectively acts as a kind of Reichhardt detector [15], that uses spatial pooling to generalize over larger regions and to deal with noise.

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 training pairs $(\boldsymbol{x}^{\boldsymbol{\alpha}}, \boldsymbol{y}^{\boldsymbol{\alpha}})$. We consider gradient based optimization for this purpose. By substituting Eqs. 2 and 4, it is easy to show that, similarly as for unconditional models [7], the gradient of the (negative) log-likelihood for each training case is the difference of two expectations:

$$-\frac{\partial L}{\partial W_{ijk}} = \sum_\alpha \left\langle \frac{\partial E(\boldsymbol{y}^{\boldsymbol{\alpha}}, \boldsymbol{h}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial W_{ijk}} \right\rangle_{\boldsymbol{h}} - \left\langle \frac{\partial E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}^{\boldsymbol{\alpha}})}{\partial W_{ijk}} \right\rangle_{\boldsymbol{h},\boldsymbol{y}} \tag{7}$$

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 [7], [20], [16], 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 [10], 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 the model 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 $W_{\cdot\cdot k}$ 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. Likewise at test time, the hidden units *de-compose* an observed transformation into its basic components by measuring correlations between inputs and outputs. 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 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 [9], for example), in which input-dependent filters $f_j = \sum_i W_{ijk}x_i$ are used to capture spatial correlations in the image. The filters are input-weighted sums of slices $W_{i\cdot\cdot}$ of the weight tensor $\boldsymbol{W}$.

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.

2.4. Gaussian outputs

The model described 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. Continuous values for the outputs are especially useful in image modelling tasks.

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 [9]: We re-define the energy as:

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

and then define the joint distribution as before (Eq. 2). Now, while the 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 distributions over output units y_j turns into Gaussians (see [9] for details):

$$y_j|\boldsymbol{x}, \boldsymbol{h} \sim \mathrm{N}(W_j^{\boldsymbol{y}} + \nu \sum_{ik} W_{ijk} x_i h_k, \nu^2) \qquad (8)$$

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 instead. As before, it is intractable to evaluate the marginal, so it is fortunate that it does not need to be to 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.

2.5. Examples

To test the model, we used a database of digitized television broadcasts. 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, *i.e.* we used only every other frame. Further details about the database can be found in [19].

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.

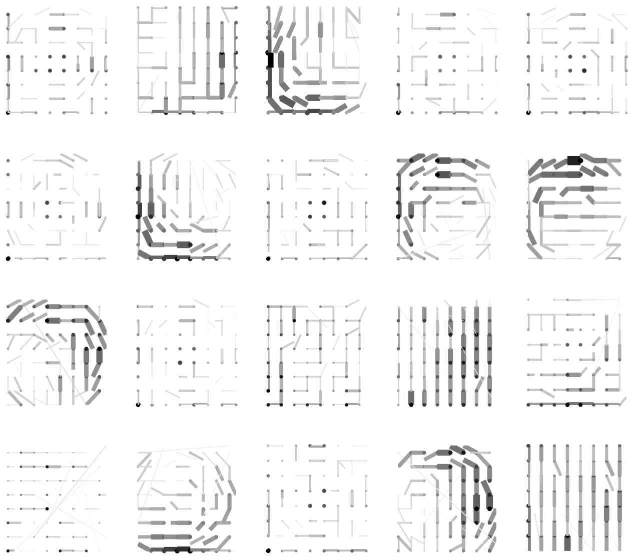
Figure 2. 20 basis flowfields learned from synthetic image transformations. The flowfields show, for each input pixel, the strengths of its positive outgoing connections, using lines whose thicknesses are proportional to the strengths.

Figure 2 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 obtain a similar plot (but poled in the opposite direction) for negative connections, showing that the model learns to locally shift edges. (See below for further results on this.)

Furthermore, the figure shows, that the model infers locality, *i.e.* 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 22×22 pixels at 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 PCA to reduce the dimensionality prior to training.

A way of visualizing the learned basis flowfields is shown in figure 3. The figure shows that the model has developed sets of local, conditional edge-filters. That is, input pixels have significant weights to output pixels that are nearby and the weights form an oriented filter. The fact that the model decides to latch mainly 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,

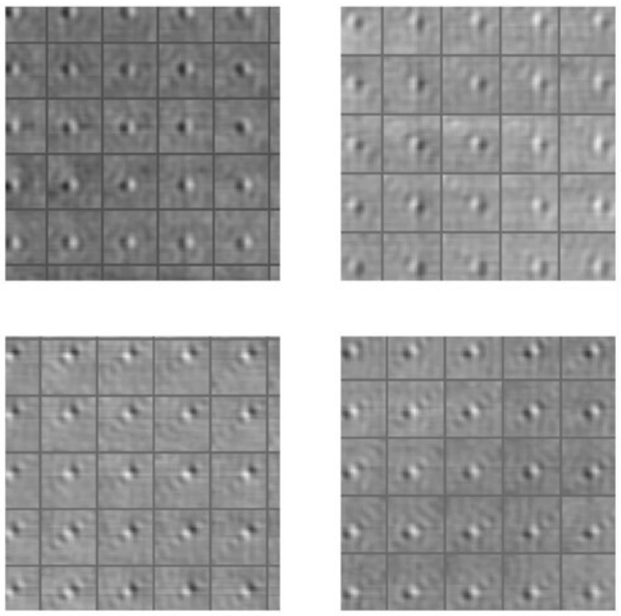

Figure 3. A visualization of parts of the learned basis flowfields W_{ijk} for four different hidden units, h_k. For each hidden unit, the input pixels in a small patch are laid out in a coarse grid. At each grid location, there is an intensity image that depicts the weights from that input pixel to 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.

that this information is learned from the database and not handcoded. No sparsity constraints were used to obtain the 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 22×22-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 shown in figure 4. The plots in the top row of the figure 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, 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[2] (bottom row of the figure). The resulting 'max-'flow-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 global motions are typical in the dataset, whose log-probability is what is being optimized during training.

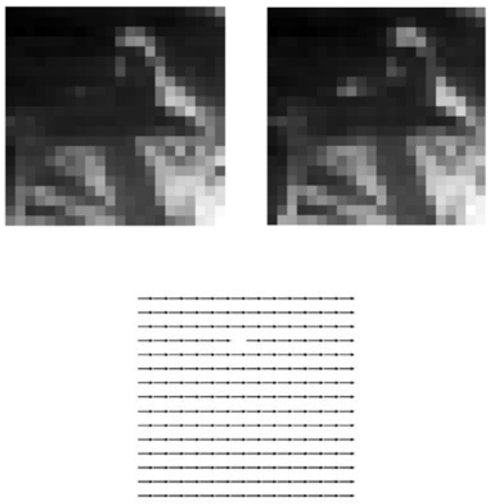

Figure 4. Typical image patch pair from the video database, and the inferred flow field. Spatial pooling facilitates generalization and noise suppression.

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 (as done in [3], for example).

2.6. Fields of Gated Experts

Until now we have 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 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

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

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 make 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

$$E(\boldsymbol{y}, \boldsymbol{h}; \boldsymbol{x}) = -\sum_s \sum_{ijk} W_{ijk} x_i^s y_j^s h_k^s, \tag{9}$$

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 [16], for the non-conditional case. A similar approach for simple conditional models has also been used by [5].

3. Application and Experiments

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 with respect to 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 [2]).

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.

3.1. Learning an invariant metric

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 as 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.

3.2. Digits

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.

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).

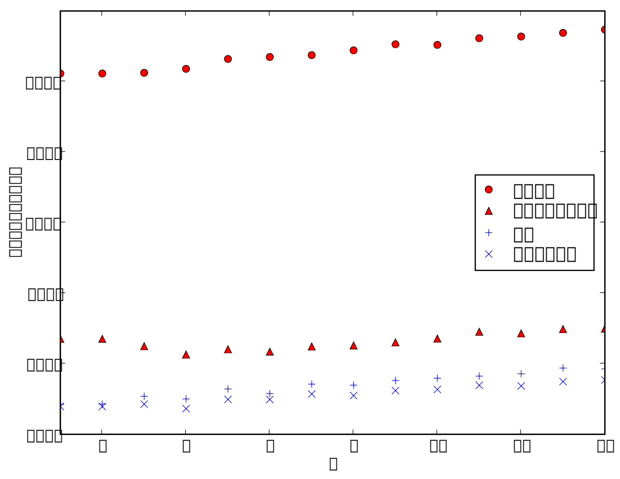
Figure 5. Error rates using k-nearest neighbors.

We 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 experiment, (ii) included in the training-set for classification in another experiment. 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 transformations.

Using 100 PCA-features to represent each image, we trained a GBM with 50 mapping units on the transformations, by predicting transformed versions of digits from the originals and vice versa. Figure 5 compares the nearest neighbor error rates on the 10000 test cases, obtained from 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 x in the three-step algorithm (previous section). In other words, we measure how well the model can transform prototypes (training cases) into the query cases.

In task (i) ('eucl' vs. 'td' in the plot) Euclidean distance 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 transformations are included in the training set, Euclidean distance improves significantly, but still does far 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.

3.3. Image transformations

Once we have a way of encoding transformations, we can also apply 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. [6] discuss a model that performs these 'image analogies' using a regression-type architecture. An interesting question is, whether it is possible to perform these analogies using a general purpose generative model of transformations.

We used a source image-pair as shown in the top row in figure 6, generated by taking a publicly available image from the database described in [4], 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 and a patch-size of 5×5 pixels. The target image is another image from the same database and is shown in the row below, along with its transformation. We obtained the transformation by performing a few hundred Gibbs iterations on the output using the original image target image as initialization (the one-step reconstruction works, too, but more iterations can improve the results a little). The blow-up shows that the model has learned to apply a somewhat regular, canvas-like structure on the output-image, as observed in the source image.

4. Conclusions

There has been surprisingly little work on the problem of learning explicit encodings of image 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 [18], 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 on multiple scales could be interesting, especially for motion-analysis. While binary hidden units amount to using a mixture of exponentially many experts, in some tasks continuous hidden units could be more suitable, in which case the model would take the form of a "continuum of experts" instead. 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.

A possible application of our model is classification based on very few training cases as suggested, for example, in [12]. Potential applications other than dealing with invariances are video compression, and discriminative, as well as temporal denoising. 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.

Standard RBMs are very 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. In contrast to previous, non-parametric approaches in this direction, such as [11], 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.

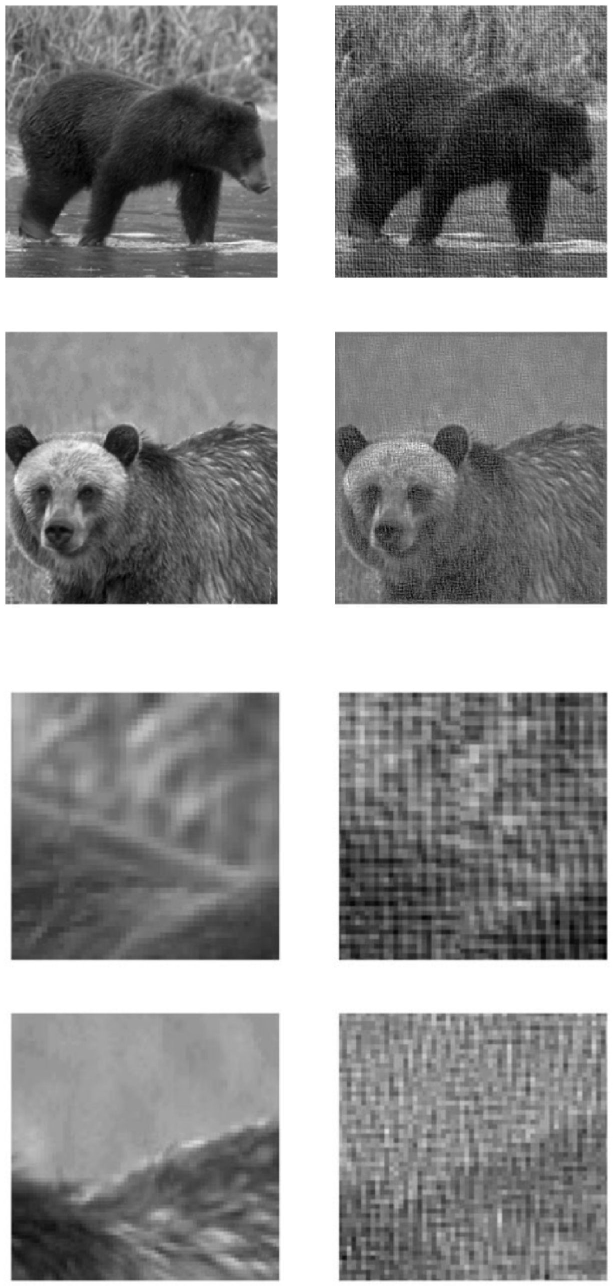
Figure 6. Image analogy. Top to bottom: Source image pair used for training, target image pair, blow-up source image pair, blow-up target image pair.

References

[1] S. J. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. Technical report, Berkeley, CA, USA, 2002.

[2] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR '05*, Washington, DC, USA, 2005. IEEE Computer Society.

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

[4] 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.

[5] X. He, R. S. Zemel, and M. A. Carreira-Perpinan. Multiscale conditional random fields for image labeling. volume 02, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[6] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *SIGGRAPH '01*, New York, NY, USA, 2001. ACM Press.

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

[8] G. E. Hinton and K. J. Lang. Shape recognition and illusory conjunctions. In *Proc. of the 9th IJCAI*, pages 252–259, Los Angeles, CA, 1985.

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

[10] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2):181–214, 1994.

[11] R. Memisevic. Kernel information embeddings. In *ICML '06*, pages 633–640, New York, NY, USA, 2006. ACM Press.

[12] E. G. Miller, N. E. Matsakis, and P. A. Viola. Learning from one example through shared densities on transforms. *CVPR*, 01:1464, 2000.

[13] B. Olshausen. *Neural Routing Circuits for Forming Invariant Representations of Visual Objects*. PhD thesis, Computation and Neural Systems, 1994.

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

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

[16] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *CVPR '05*, Washington, DC, USA, 2005. IEEE Computer Society.

[17] T. J. Sejnowski. Higher-order boltzmann machines. In *AIP Conference Proceedings 151 on Neural Networks for Computing*, pages 398–403, Woodbury, NY, USA, 1987. American Institute of Physics Inc.

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

[19] 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, 1998.

[20] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.