Gated Autoencoders with Tied Input Weights

Droniou Alain ALAIN.DRONIOU@ISIR.UPMC.FR
Sigaud Olivier OLIVIER.SIGAUD@UPMC.FR
Institut des Systèmes Intelligents et de Robotique (ISIR)
Université Pierre et Marie Curie - CNRS7222
4 Place Jussieu, 75005 Paris, France

Abstract

The semantic interpretation of images is one of the core applications of deep learning. Several techniques have been recently proposed to model the relation between two images, with application to pose estimation, action recognition or invariant object recognition. Among these techniques, higher-order Boltzmann machines or relational autoencoders consider projections of the images on different subspaces and intermediate layers act as transformation specific detectors. In this work, we extend the mathematical study of (Memisevic, 2012b) to show that it is possible to use a unique projection for both images in a way that turns intermediate layers as spectrum encoders of transformations. We show that this results in networks that are easier to tune and have greater generalization capabilities.

1. Introduction

Recognizing actions from images is an important challenge for developmental robotics (Fitzpatrick et al., 2003; Montesano et al., 2008). Gated neural networks can lead to state-of-the art performances in action recognition (Taylor et al., 2010), and are very good at encoding relation between images (Memisevic & Hinton, 2007; Memisevic, 2012a).

The performances of gated neural networks come from their multiplicative interactions between different layers. It actually allows the network to uncouple the representation of a "dictionary" and a representation of the plausible transformations of its elements. In the particular case of the encoding of relation between images, it allows the network to focus on the transformation between two images without spending computational resources on the representation of their content: the elements of the dictionary are the images themselves, which are provided as inputs. In this case, the role of the network is to discover an efficient representation of correlations between images, which is encoded in a layer commonly referred to as "mapping units".

Recently, (Memisevic, 2012b) proposed a mathematical framework for gated autoencoders which links energy-based models, such as higher-order Boltzmann machines (Memisevic & Hinton, 2010), with relational autoencoders. This framework shows that classical gated autoencoders learn pairs of filters which are related by the transformation to detect. The projection of both images on these pairs creates a correlation peak when the transformation between the two images is consistent with the transformation encoded in a pair of filters. Moreover, for each transformation to be detected, two pairs of filters are learned in order to span the subset of images: to be able to detect a transformation independently of the content of the images, the two pairs are in quadrature such that if the images are orthogonal to the first pair of filters, the transformation will be detected by the other pair. A factor layer computes the correlation between the projections of images on each pair of filters, and a mapping layer pools over the factor units corresponding to quadrature pairs to obtain a content independent detection of the transformation. This implies that the number of filters and mapping units has to grow with the desired accuracy of transformation detection, since the number of mapping units grows linearly with the step size between two transformations to be distinguished. This can lead to prohibitive network sizes for real world applications such as action recognition, especially when the number of filter pairs increases, since each filter adds one weight per image pixel.

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

Furthermore, Memisevic's work shows that commuting transformations, like translations or rotations, can share the same set of filters. In this case, the detection of a specific transformation requires the detection of an angle between the projection of the first image on the filters in quadrature and the projection of the second image on the same filters. However, this raises the aperture problem if we detect the angle after normalizing the projections, as suggested by Memisevic: this can lead to false or very inaccurate detection of transformations when the projections of the images are close to zero.

In this work, we show that projections normalization is not mandatory. Thanks to this approach, mapping units are no longer transformation specific and we show that the mapping layer represents a discretization of the spectrum of the transformations instead of a discretization of the transformations themselves. This reduces the size needed for a correct representation of transformations, from $n \times n$ matrices to n eigenvalues.

2. Gated autoencoders

In this section, we briefly recall the mathematical analysis of gated autoencoders (a.k.a. relational autoencoders) from (Memisevic, 2012b).

Let us consider two images x and y in the form of a vector ($x, y \in \mathbb{R}^n$) related by a transformation L, such that

$$y = Lx \tag{1}$$

where L is an orthogonal matrix ($LL^\top = I$, I being the identity matrix). In particular, orthogonal matrices cover all the permutations of pixels and these transformations can be roughly seen as "constant information" transformations where all the pixels of one image can be fully predicted from the pixels of the other image.

Orthogonal matrices can be diagonalized by

$$L = UDU^\top \tag{2}$$

where D is the diagonal matrix containing eigenvalues which are all complex numbers of module 1, and U contains the eigenvectors. When two transformations of the same space L_1 and L_2 commute, they share the same eigenvectors and their factorization differs only by the diagonal matrix. Remarkably from (2), since L is a real matrix, we have $U^{-1} = U^\top$ and (1) can be rewritten

$$U^\top y = DU^\top x. \tag{3}$$

It then appears that the eigenvalues can be retrieved by detecting the rotation angles between the projections of x and y[1]. Memisevic showed that the inner product between normalized projections directly provides the cosine of the angle, but the normalization of these projections is unstable when they are too close to zero, leading to false detection of transformations.

(Memisevic, 2012a;b) reformulates the problem as a detection task consisting in learning input and output filters U and V, such that V incorporates D and U. Then, Eq. 3 becomes

$$U^\top y = V^\top x \tag{4}$$

and the correlation between input and output filters is a detector whose value depends on the transformation and the content of the images, through their projections on U and V. To have a content independent representation, it is possible to pool over a set of detectors which represent the same transformation but span the space of images (they are in "quadrature"). In these architectures, a factor layer computes the correlations between projections, while a mapping layer pools over the factor layer to obtain the transformation specific detectors.

3. Encoding relationship with tied input weights

The approach described in Section 2 is suboptimal since it requires several pairs of filters to represent each specific transformation. The idea behind our work is to use directly Eq. (2) and make the factor layer represent the diagonal matrix D, while filters represent the matrix U. This leads to a simpler network, since there is no more distinction between input and output filters and the filters are shared between all commuting transformations.

3.1. Theoretical description

Recalling Eq. (3) and the fact that D contains complex values of module 1, all the information about D can be retrieved through the cosine and sine of the angle between projections of x and y, that we note generically u_x and u_y. If we see these complex values as 2D vectors, the inner product $u_x.u_y$ and the magnitude of the cross product $u_x \times {u_y}^2$ provide cosine and sine up to a multiplicative factor $||u_x||.||u_y||$ which is the same for both values:

$$\begin{aligned} u_x.u_y &= ||u_x||.||u_y||.cos(u_x, u_y) \\ u_x \times u_y &= ||u_x||.||u_y||.sin(u_x, u_y). \end{aligned} \tag{5}$$

[1] A multiplication by a complex of module 1 is actually equivalent to a rotation in the complex plane

[2] The magnitude corresponds to the cross product without the unit vector.

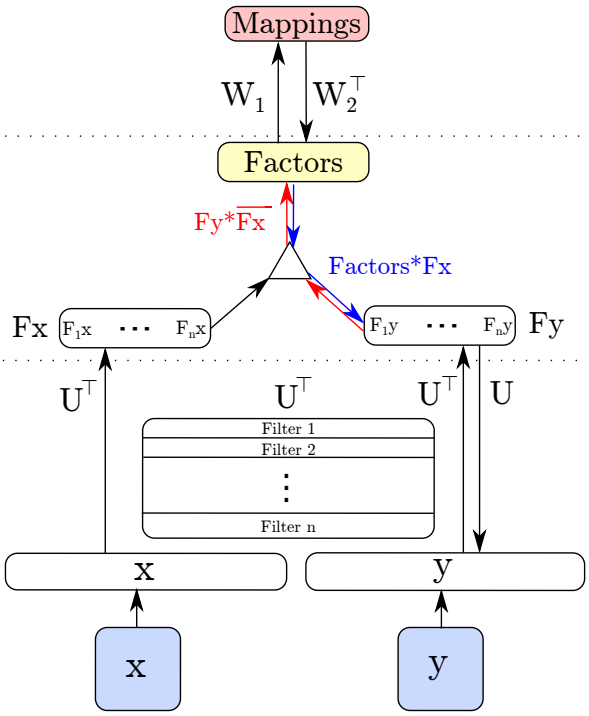

Figure 1. Gated autoencoder with tied input weights.

These equations are sufficient to retrieve the angle modulo 2π, since their quotient gives the tangent of the angle. However, taking this quotient would lead to the same problem as normalization, with very instable values when the projections (i.e. norms of u_x and u_y) are close to zero. It is much better to keep cosine and sine values apart, so that small values simply indicate that the corresponding eigenvectors are not relevant to describe the considered transformation. Considering complex numbers, the inner product is given by $Real(u_y\overline{u_x})$ and the magnitude of cross product is given by $Imag(u_y\overline{u_x})$ where $\overline{\cdot}$ is the conjugate and $Real$ and $Imag$ are respectively the real and imaginary parts of a complex number. To sum up, all the information is contained in $u_y\overline{u_x}$[3].

Our factor layer is thus given by

$$f = U^\top y * \overline{U^\top x} \tag{6}$$

where $*$ denotes the element-wise multiplication. We can then use the mapping layer of classical relational autoencoders to compress the factor layer into a lower dimensionality representation.

We train our network as classical gated autoencoders, where the goal is to minimize the error of the reconstruction of the second image from the first. To do that, each image is filtered by a stack of filters U, which project the images on the factor space, which is then projected on the mapping layer. A backward pass through the network then produces the reconstruction of the output image (Fig. 1). We also adopt the denoising approach, in which inputs are corrupted with some noise before flowing through the network, to avoid overfitting (Vincent et al., 2010). In our case, we use a zeromask noise, which sets a fraction of the pixels (typically 30%) to zero and lets the others unaltered. We denote $\hat{x}$ the corrupted version of x. This constraint is all the more important than sparsity-based regularizations (such as (Lee et al., 2008)) are not well-suited with a spectrum representation.

The mapping layer is given by

$$m = \sigma\left(W_1\hat{f} + bias_{\text{mappings}}\right) \tag{7}$$

where $\hat{f}$ is the factor layer (Eq. 6) obtained from the corrupted version of x and y and σ is the activation function of rectified linear units (a.k.a. softplus units): $\sigma(x) = log(1+exp(x))$. Note that we use rectified linear units where classical gated autoencoders use sigmoid units. In fact, since classical gated autoencoders act as "transformation specific detectors" and pool over a set of detectors for each transformation, the mapping units represent the presence or the absence of a given transformation and a "probability" unit is a good choice. In our case, we want to represent the spectrum of the transformation, such that linear units are better suited than sigmoid units.

The reconstruction is then

$$r = U(W_2^\top m * U^\top\hat{x}) + bias_{\text{output}}. \tag{8}$$

The reconstruction error used for the gradient descent is finally given by

$$error = ||y - r||^2. \tag{9}$$

Note that two weight matrices W_1 and W_2 are used between the factor and mapping layers. This is useful since the reconstruction given by (8) is not strictly correct from our mathematical description. In particular, it can be seen closer to $r = ||U^\top x||^2UDU^\top x$ (because $u_y\overline{u_x} = d||u_x||^2$) whereas we expect $y = UDU^\top x$ (Eq. 1 and 2). This would be the case in particular if $W_1 = W_2$, but untying these weights allows the network to scale the reconstructed activation of the factor layer differently from its bottom-up activation. It also lets the network cancel out factor values which correspond to irrelevant eigenvectors for a given transformation in the case where the learning set is composed of several classes of transformations.

[3]From (3), we deduce $u_y\overline{u_x} = d||u_x||^2$.

3.2. Practical implementation

Our mathematical description involves complex numbers, whereas neural networks generally work on real numbers and very few studies have been carried on more general mathematical frameworks (see for instance (Baldi, 2012; Baldi et al., 2012)). In particular, our study involves multiplications of complex numbers. When working with real numbers, we have to split real and imaginary parts of each filter into filter pairs, as mentioned in (Memisevic, 2012b). However, the element-wise multiplication between projections of both images, as usually performed in gated autoencoders, makes impossible to retrieve the complex multiplication. In previous gated autoencoders, this was hidden by the fact that only specific detectors were learned, so that only the cosine of the angle was necessary and a simple inner product was sufficient. In our case, we need to retrieve both the cosine and sine of the angle, and we also need the cross-product. Otherwise, it would be impossible to differentiate symmetric transformations, such as rotations of θ and $-\theta$. One way to deal with this issue consists in making a matrix multiplication between both projections, which would create all possible two-terms products, but this would lead to a huge factor layer, with many weights to project it on the mapping layer. Moreover, most of these coefficients would be useless, since they involve multiplications between unrelated filters. This also prevents mini-batch learning optimizations through matricial representation of multiple training cases: one training case cannot flow through the network as one column of a matrix, since the factor layer would require a matrix for each case.

Our proposed solution is to put prior knowledge on complex multiplication inside the connectivity of the network. To do that, we artificially duplicate the factor layer: the first half involves term-to-term multiplications, while the second half is "crossed" so that term-to-term multiplications correspond to the multiplication between real and imaginary part of each filter. In other words, we shape the network with the sufficient structure to learn filters corresponding to our mathematical description. "Crossing" the second half of the factors simply consists in duplicating factors from one image, while the factors from the other image are multiplied by a block-diagonal matrix B composed of $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ blocks.

We also use a pooling matrix to tie the learning process of pairs of filters, as mentioned in (Memisevic, 2012a), and retrieve directly the values of inner and cross products. This pooling matrix is also useful to restrict the final factor layer size to the one obtained before our artificial duplication of factors. The whole process is illustrated in Fig. 2.

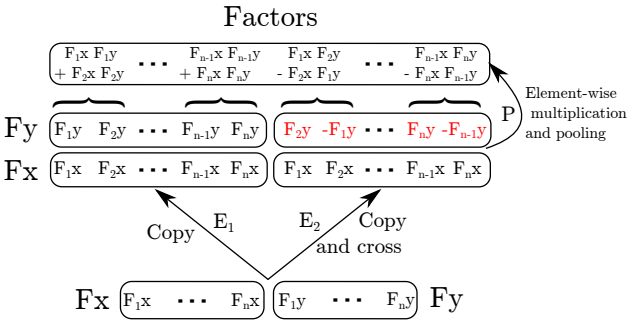

Figure 2. The real-valued implementation of our algorithm involves some hard-coded operations to simulate complex-valued multiplications. These operations are computed by multiplications with constant matrices (see Section 3.2).

Let P_l be the pooling matrix of size $l \times 2l$

$$P_l = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 0 & \dots \\ & & & \ddots & & & \end{pmatrix},$$

I_l the identity matrix of size l, B_l the block-diagonal matrix B defined above with size l, R_l a reordering matrix

$$R_l = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & \dots & -1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & \dots & 0 & -1 & 0 & \dots \\ & & & & \vdots & & & \end{pmatrix}$$

and E_1, E_2, and E_3 duplicating matrices of size $2l \times l$ where l is the size of the factor layer

$$E_1 = \begin{pmatrix} I_l \\ I_l \end{pmatrix}, \; E_2 = \begin{pmatrix} I_l \\ B_l \end{pmatrix}, \; E_3 = \begin{pmatrix} R_l \\ B_l R_l \end{pmatrix}.$$

Equations (7) and (8) can finally be rewritten:

$$\begin{aligned} m &= \sigma\left(W_1 P((E_1 U^\top \hat{x}) * (E_2 U^\top \hat{y})) + bias_{\text{mappings}}\right) \\ r &= UP((E_3 W_2^\top m) * (E_1 U^\top \hat{x})) + bias_{\text{output}} \end{aligned} \tag{10}$$

Note that E_3 could be incorporated in W_2 during the learning process rather than being defined a priori. In that case, W_2 should have twice more rows than W_1.

3.3. Relation with quadrature pairs

The notion of "generalized quadrature pairs" was introduced by (Bethge et al., 2007), to highlight the fact

that any orthogonal transformation L has a block diagonal representation $W^\top LW = G \in \mathbb{R}^{m\times m}$ such that each block is at most two-dimensional, and each two-dimensional block belongs to $SO(2,\mathbb{R})$ (special orthogonal group). Therefore, the corresponding pairs of basis functions are referred to "generalized quadrature pairs".

Remarkably, if we note $L = UDU^\top$ the eigendecomposition of L (which involves complex numbers), we have $G = MDM^\top$ where M is a block diagonal matrix with blocks $\frac{\sqrt{2}}{2}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$. Then, $L = UDU^\top = WGW^\top = UM^\top GMU^\top$, which leads to $W^\top UM^\top G = GW^\top UM^\top$. Thus, G and $W^\top UM^\top$ commute, so they share the same eigenvectors. Since $D = M^\top GM$, the eigenvectors of G are the columns of M, and $W^\top UM^\top M = M\Theta$ where Θ is a diagonal matrix containing the eigenvalues of $W^\top UM^\top$ (which are all complex numbers of module 1 by orthogonality), leading to $U = WM\Theta$.

Finally, the real and imaginary part of each complex filter in U correspond to a quadrature pair of W, up to a rotation in the complex plane (matrix Θ).

4. Experiments

In this section, we present some experimental results to show the effectiveness of our approach.

4.1. Experimental setup

For all our experiments, we use 13x13 pixels images consisting in random dots, whose values are randomly drawn from a normal distribution and independent from one another. We then apply translations combining uniformly random +/-3 pixels shifts horizontally and vertically, or rotations between -50 and +50 degrees. Pixels of the output image not covered by the input image are then also drawn randomly according to the same law. For each experiment, we create a training set of 100 000 pairs of images, equally distributed among the considered transformations. We measure the error as the arithmetic mean of $||y - r||_2^2$ where r is the reconstruction of y (Eq. 10).

We use the network presented in the previous section with different sizes for each layer and we decrease the learning rate during the process. We use the theano[4] python library (Bergstra et al., 2010), and our code is an extension of Memisevic's code of gated autoencoders, available online[5]. This code is also used for comparison between our approach and a classical gated autoencoder in Sections 4.3 and 4.4. In this section, we will refer to Memisevic's implementation of classical gated encoder as "CGA". Since our network uses softplus units at the mapping layer, instead of sigmoid units for CGA, we also test a modified version of CGA with softplus units, which we call "CGA-softplus".

[4]http://deeplearning.net/software/theano/

[5]http://learning.cs.toronto.edu/ rfm/code/rae/index.html

Table 1. Common parameters for all the experiments, for both CGA and our algorithm.

Parameter	Value
Corruption type	Zero mask
Corruption level	0.3
Weight decay	0
Minibatch size	100
Input size	13×13 pixels
Learning rate	$\frac{0.005}{max(1, floor(\text{epoch} * 0.1))}$

We keep some parameters constant for all the experiments. They are given in Table 1. The initial learning rate is chosen such that no instability of the mean reconstruction error is visible during learning. For other parameters, the values are those by default in Memisevic's code, except for the corruption level that we fixed at 30% instead of 50%.

Finally, we recall that the mathematical study is valid only for "constant information" transformations. In practice, this condition is rarely met: for a translation for instance, pixels at the edges cannot be deduced from the previous image. To be closer to the theoretical conditions, we apply a Gaussian mask on the images when computing the reconstruction error, such that edge pixels have a smaller weight than center ones.

4.2. Experiment 1: Factor layer as spectrum representation of transformation

In this experiment, we want to test the validity of our mathematical approach, to see if the network behavior meets our expectations.

We use a property of orthogonal transformations, that is $LL^\top = I$, i.e. $L^\top = U\overline{D}U^\top$. Thus, by testing inverse transformations, like rotations of angle $+\theta$ and $-\theta$, we expect the corresponding factor values to be conjugate, i.e. their inner products to be the same while cross products to have opposite signs. Therefore, we train a network with 400 units on the factor layer and 40 units on the mapping layer on rotations and we compare factor values for opposite rotations. We average the result over 100 rotations of different images for each angle. As expected, we can see in

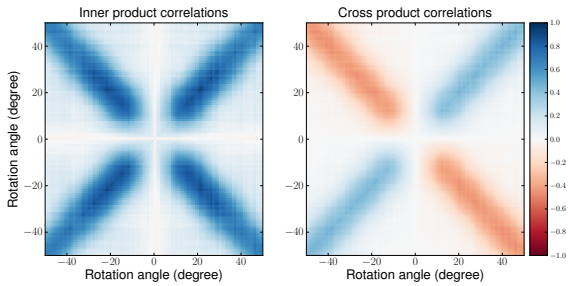

Figure 3. Mean correlations of the activation of units in the factor layer for rotations between -50 and 50 degrees (represented by the covariance matrix). We can see that, as expected, the network learns both an inner product part and a cross product part, the former being correlated between opposite rotations, while the latter is anticorrelated.

Fig. 3 that values corresponding to the inner products are correlated, while values corresponding to the cross products are anti-correlated.

Our network is thus learning a representation consistent with our mathematical description, even if experimental conditions do not meet exactly our mathematical assumption of orthogonal transformations, given noise and border effects.

4.3. Experiment 2: Filters are shared between commuting transformations

In this section, we want to demonstrate the higher generality of the learned features compared to CGA described in (Memisevic, 2012a). This relies on the fact that learned filters are shared between all commuting transformations, whereas in the other approach, each filter is specific to a given transformation.

To do that, we train the same network as above on a sparse subset of rotations, of -50,-40,-30,..., 40 and 50 degrees. Then, we test the network on every rotation of integer angles between -50 and 50 degrees. As before, we average our results on 100 rotations of different images for each angle.

Fig. 4 represents the distance between mapping layer activation for each rotation. CGA tends to create clusters around rotations in the training set, whereas our network produces a smoother representation: it interpolates its representations, whereas CGA acts more like a nearest neighbor algorithm. This effect is less visible for CGA-softplus. This can be explained because sigmoid units tend to force the activation to be 0 or 1 (to be "transformation specific" detectors), whereas softplus units allow smoother variations. However, the discriminative power of our network is better than both versions of CGA: out of the

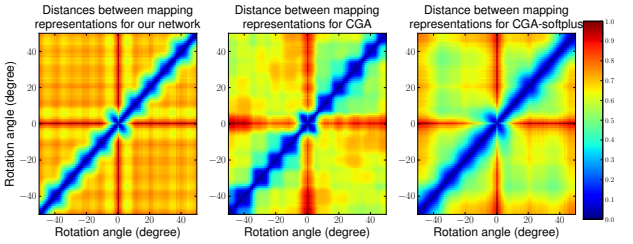

Figure 4. Distances between mean activation of the mapping layer for rotations between -50 and 50 degrees with a network trained with only 11 different rotations (see text for details). Distances have been normalized between 0 and 1. Our network produces a smoother representation of rotations compared to CGA and has a higher discriminative power: the "block" effect around rotations in the training set, which leads the network to assimilate all the rotations in the block, is less visible and distances out of the main diagonal are bigger.

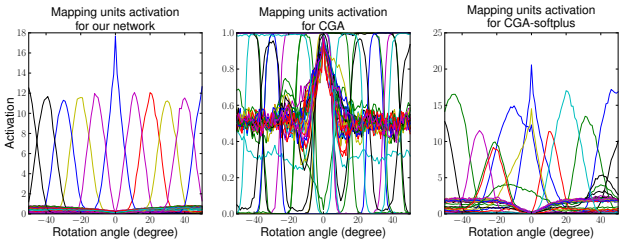

Figure 5. Activation of mapping units for rotations between -50 and 50 degrees with a network trained on 11 different rotations. Our network learns a simpler representation compared with CGA and CGA-softplus. In particular, one mapping unit specializes for each rotation in the training set.

main diagonal, distances are bigger and the transition is sharper.

Mapping layer activations in our network and in CGA are compared in Fig. 5. The representation learned by our network is simpler: one unit specializes for each rotation in the dataset. Since we have only 11 different rotations in the dataset for 40 mapping units, 29 units are not used and their activation keeps close to 0. This raises one question: how relevant for generalization is the representation learned by our network?

First, we can assume that the learned weights for each unit correspond to the spectrum of the rotation for which its response is maximum. Then, the network approximates intermediate rotations between two consecutive learned rotations by a linear mixing of their spectrum. If rotations in the training set are not too far from each other, this is a good approximation. Let A and B be two commuting transformations and a and b their eigenvalues corresponding to eigenvector v. We

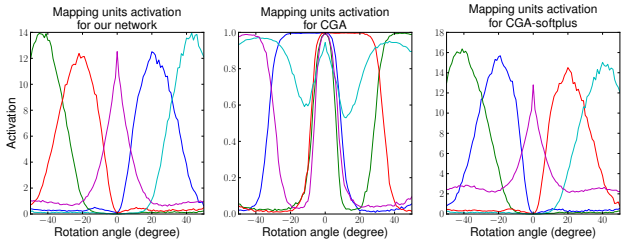

Figure 6. Activation of mapping units for rotations between -50 and 50 degrees. Compared to Fig. 5, there are less units than rotations in the training set. In this case, mapping units span regularly the set of rotations.

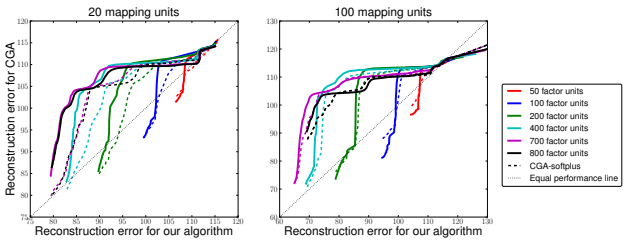

Figure 7. Learning curves for our algorithm and CGA, both trained for 500 epochs. We represent the evolution of the reconstruction error between both algorithms. Temporal course happens from upper right corner towards bottom left corner. We can see that for small factor layers, CGA is better (the end point is below the principal axis), but our algorithm converges faster and becomes better for bigger factor layers (see the text for explanation). However, we must recall that for a given factor layer size, CGA has twice more input-to-factor weights than our algorithm. When we compare for a given number of weights, our algorithm is better (see Fig. 8).

consider T the average transformation between A and B, i.e. $T = (AB)^{\frac{1}{2}}$ where we define the matrix square root by $M^{\frac{1}{2}}M^{\frac{1}{2}} = M$. In particular when M is diagonalizable, $M = VDV^{-1}$, then $M^{\frac{1}{2}} = VD^{\frac{1}{2}}V^{-1}$ and the eigenvalues of $M^{\frac{1}{2}}$ are the square roots of those of M. In our case, the eigenvalues of T are thus the geometric mean of the eigenvalues of A and B: $t = \sqrt{ab}$. Finally, approximating the geometric mean by the arithmetic mean is a good approximation when both numbers are not too far from each other (which is the case for the eigenvalues of nearby orthogonal matrices, by continuity): if $a = c + \delta$ and $b = c - \delta$, the first order error is $\frac{\delta^2}{2c}$.

Fig. 6 shows what happens when there are less mapping units than transformations in the training set. We use the same dataset as above, but with only 5 mapping units. In this case, the mapping units span regularly the set of transformations for the three algorithms: since the resources are very constrained, each unit is forced to represent a different subset of transformations. These results illustrate the fact that our network is easier to tune, since an oversized mapping layer has a lower impact on its generalization capabilities than for CGA.

4.4. Experiment 3: Comparison between CGA and our tied input weights autoencoder

Finally, we compare the performance of our network with the one described in (Memisevic, 2012a). We train the networks for 500 epochs on a set of rotations and translations, and measure their reconstruction error on test images spanning the whole set of transformations we used for training (10 000 pairs of images equally distributed). Given the proximity between both algorithms, we compare their performances with the same parameters, especially for the learning rates, the mini-batch size and the input corruption level (see Table 1).

Fig. 7 shows the learning curves of both algorithms for different sets of parameters. Our algorithm converges faster and has a lower final error than CGA when the number of factor units is large enough. In contrast, CGA is better for small factor layers. This can be surprising given our study, which states that filters are shared between commuting transformations. However, when there are not enough factor units to represent all the eigenvectors of a transformation, this is equivalent to assigning a null eigenvalue to the missing eigenvectors, while eigenvalues of orthogonal transformations are values of module 1. Moreover, we must recall that for a given size of factor layer, CGA has twice more input-to-factor weights providing more flexibility. This explains why our algorithm makes worse estimations for small factor layers. On the other hand, we can see that the limit predicted by the mathematical description, which is reached when the size of the factor layer is equal to the number of eigenvectors for every transformations, is confirmed by our experiments. With two classes of transformations, we expect a limit for $2 \times 2 \times 13^2 = 676$ factor units[6].

Fig. 8 plots the mean reconstruction error on the test set for the algorithms. We can see that CGA is subject to a severe degradation of performances when the number of factor units increases, which is not the case for our algorithm (the performance only slightly de-

[6]Each transformation is represented by a 169×169 matrix, and has 169 eigenvectors. In our architecture, each eigenvector is represented by two factor units (real and imaginary parts).

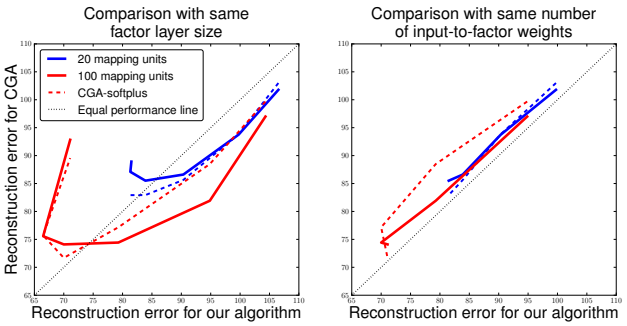

Figure 8. Mean reconstruction error on the test set for both algorithms with 50, 100, 200, 400, 700 and 800 factor units. Left: comparison with the same factor layer size for both algorithms; right: comparison with the same number of input-to-factor weights. For a given number of free parameters, our algorithm outperforms CGA (right).

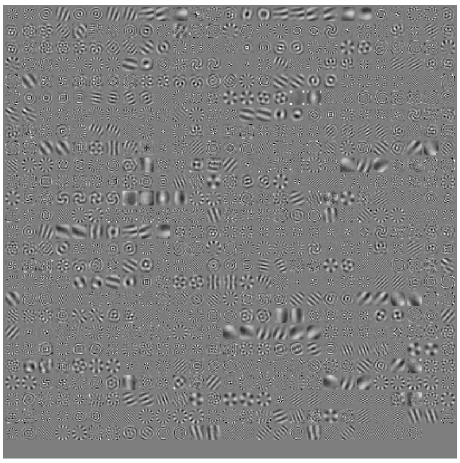

Figure 9. Features learned at the factor level by our algorithm on a dataset which contains rotations and translations. The network has 700 factor units and 100 mapping units.

creases for 100 mapping units). This observation is coherent with the fact that our network generalizes better.

In particular, our network is easier to design since an oversized factor layer has a smaller impact on performance compared with CGA, and there are less dependences between factor and mapping layers sizes (Fig. 8).

Fig. 9 shows the filters learned by our network. As expected, most filters come by pairs which represent the real and the imaginary part of one complex filter. They are similar to the ones learned by a CGA (see for instance (Memisevic, 2012a)).

5. Conclusion and future work

In this work we presented a new architecture for relational autoencoders, which achieves better performances with respect to previous methods, in particular from the generalization point of view, while having less parameters to learn. It relies on a complex-valued reformulation of classical gated networks.

Our network differentiates from previous gated autoencoders since both images are projected on the same subspace, through the same matrix of weights U. There is thus only one weight matrix shared between inputs at different timesteps. This makes our network closer to classical recurrent networks than usual gated networks which consider either a sliding window of a time dependent input signal or pairs of related inputs and associate each input with a different weight matrix (Le et al., 2011; Memisevic, 2012a; Susskind et al., 2011; Sutskever & Hinton, 2007; Taylor et al., 2011; Baccouche et al., 2011). This opens the way to a simple recurrent gated autoencoder, where the input consists of only one image. In particular, we believe that this could be useful for motion analysis with a moving camera, a situation often encountered in robotics, for instance. Actually, motions of the camera induce images transformations like translations and rotations, which can be encoded by a network similar to the one described in this paper. A multiplicative interaction at the mapping layer, between visual inputs and an action-based encoding of the camera's motion (or of the gaze position, as in (Larochelle & Hinton, 2010; Denil et al., 2012)) could achieve better performance and robustness in motion recognition.

One of the obvious extensions of our work is to handle more realistic image sizes, by using for instance convolutional techniques, to use the network on real videos streamed from cameras. Like other gated networks, the algorithmic complexity of our algorithm is actually quadratic in the number of pixels. This makes these algorithms unscalable directly to bigger images and convolutional techniques are one way to keep computational complexity under control. Furthermore, convolutional techniques are well suited to handle some invariances, like the position of a moving object in a video, and provide greater factorization possibilities, for instance when several objects are moving in different ways on the same video.

Finally, unsupervised clustering algorithms, such as self-organizing maps, could be used to cluster autonomously the different classes of transformations and extract a more semantic discretization of the observed transformations.

Acknowledgments

This work is supported by the French ANR program (ANR 2010 BLAN 0216 01), more at http://macsi.isir.upmc.fr

References

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., and Baskurt, A. Sequential Deep Learning for Human Action Recognition. In A.A. Salah, B. L. (ed.), *2nd International Workshop on Human Behavior Understanding (HBU)*, Lecture Notes in Computer Science, pp. 29–39. Springer, 2011.

Baldi, P. Autoencoders, Unsupervised Learning, and Deep Architectures. *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012.

Baldi, P., Forouzan, S., and Lu, Z. Complex-Valued Autoencoders. *Neural Networks*, 33:136–147, 2012.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of SciPy*, 2010.

Bethge, M., Gerwinn, S., and Macke, J. H. Unsupervised learning of a steerable basis for invariant image representations. In *Proceedings of SPIE Human Vision and Electronic Imaging*, 2007.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. Learning where to Attend with Deep Architectures for Image Tracking. *Neural Computation*, 2012.

Fitzpatrick, P., Metta, G., Natale, L., Rao, S., and Sandini, G. Learning about objects through action – initial steps towards artificial cognition. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3140–3145, 2003.

Larochelle, H. and Hinton, G. Learning to combine foveal glimpses with a third-order Boltzmann machine. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23*, pp. 1243–1251. 2010.

Le, Q. V., Zou, W. Y., Yeung, S. Y., and Ng, A. Y. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR 2011*, pp. 3361–3368. IEEE, June 2011.

Lee, H., Ekanadham, C., and Ng, A. Sparse deep belief net model for visual area V2. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 873–880. MIT Press, Cambridge, MA, 2008.

Memisevic, R. On multi-view feature learning. In *ICML*, 2012a.

Memisevic, R. Learning to relate images: Mapping units, complex cells and simultaneous eigenspaces. *ArXiv e-prints*, 2012b.

Memisevic, R. and Hinton, G. Unsupervised Learning of Image Transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

Memisevic, R. and Hinton, G. E. Learning to Represent Spatial Transformations with Factored Higher-Order Boltzmann Machines. *Neural Computation*, 22(6):1473–1492, 2010.

Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, Feb. 2008.

Susskind, J., Memisevic, R., Hinton, G., and Pollefeys, M. Modeling the joint density of two images under a Variety of Transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Sutskever, I. and Hinton, G. E. Learning Multilevel Distributed Representations for High-dimensional Sequences. *Proceeding of the Eleventh International Conference on Artificial Intelligence and Statistics*, 2007.

Taylor, G. W., Fergus, R., LeCun, Y., and Bregler, C. Convolutional learning of spatio-temporal features. In *ECCV'10*, pp. 140–153, September 2010.

Taylor, G. W., Hinton, G. E., and Roweis, S. T. Two Distributed-State Models For Generating High-Dimensional Time Series. *J. Mach. Learn. Res.*, 12: 1025–1068, 2011. ISSN 1532-4435.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010. ISSN 1532-4435.