Learning Deep Neural Networks for High Dimensional Output Problems

Benjamin Labbé, Romain Hérault, Clement Chatelain

Learning Deep Neural Networks for High Dimensional Output Problems

Benjamin Labbé Romain Hérault
Clément Chatelain
LITIS EA 4108, INSA de Rouen, Saint Étienne du Rouvray 76800, France
name.lastname@insa-rouen.fr

Abstract

State-of-the-art pattern recognition methods have difficulty dealing with problems where the dimension of the output space is large. In this article, we propose a new framework based on deep architectures (e.g. Deep Neural Networks) in order to deal with this issue. Deep architectures have proven to be efficient for high dimensional input problems such as image classification, due to their ability to embed the input space. The main contribution of this article is the extension of the embedding procedure to both the input and output spaces in order to easily handle high dimensional output problems. Using this extension, inter-output dependencies can be modelled efficiently. This provides an interesting alternative to probabilistic models such as HMM and CRF. Preliminary experiments on toy datasets and USPS character reconstruction show promising results.

1. Introduction

High dimension output tasks is a challenging problem in both pattern recognition and machine learning communities. It is concerned with problems where the target $\mathbf{y}$ is a vector in $\mathbb{R}^n$ and $n \gg 1$ (instead of a scalar in $\mathbb{R}$ for a standard regression problem). For instance, in an image segmentation problem, $\mathbf{y} \in [1. \, . \, .255]^{n \times m}$, where $n \times m$ is the size of the image. When dealing with multiple outputs, most of the time, strong relations between these outputs can be observed. For example, the label of a pixel strongly depends on the labels of its neighbors. In order to exploit these relations, a global decision overall outputs should be taken instead of multiple local decisions.

In the literature, numerous approaches have been successfully applied on real-world sequence signals, such as handwriting recognition [6], voice recognition [10], or information extraction in text [2]. These approaches are mainly based on probabilistic Hidden Markov Models, where relations between the outputs are learned. One may note that in the HMM framework, the observations are supposed to be independent, which is wrong in almost all applications. More recently, the Conditional Random Fields (CRF), derived the HMM to a discriminative model. CRF have been proposed to overcome this problem by modeling the conditional probability $p(\mathbf{x}|\mathbf{y})$ instead of the joint probability $p(\mathbf{x}, \mathbf{y})$.

Although efficient on sequences, only few methods, usually based on an extension of HMM or CRF, have been applied on two-dimensional signals. HMM can be adapted to 2D signals through either pseudo-2D HMM or Markov Random Field (MRF). [9] provides an example of MRF for document image segmentation. CRF have also been generalized for 2D-signals with application to diagram interpretation [11]. To the best of our knowledge, there is no approach for signal dimension higher than 3 [12]. It is the limitation of these kind of approach. The reason why few probabilistic methods are applied on 2D-signals is mainly due to their decoding complexity. Even if some sub-optimal decoding methods have been proposed such as HCF or ICM, this is still an open issue. Nowadays, probabilistic methods are the state-of-the-art learning methods for tasks such as signal and image segmentation. Let us also mention the kernel dependency estimation [13] and structured output SVM approaches [3]. These approaches are based on a kernel joint projection which evaluate the co-occurrence probability of an observation $\mathbf{x}$ and a complex label $\mathbf{y}$. Although these approaches handle complex output spaces, they suffer from the pre-image problem. They can easily give an appropriate image in a kernelized label space for a given $\mathbf{x}$, but the label itself is hard to predict explicitly.

In this paper, we assume that other machine learning methods such as neural network are able to handle high dimension problems, whatever the dimension of the signal, and whatever the topology of the input or the output. Deep Neural Network (DNN) with pre-training [7, 1] or regularized learning process [14] have achieved good performance on difficult high dimensional input tasks where Multi Layer Perceptron (MLP) seems to failed before. These methods try to find a better representation of the input space in an unsupervised manner to ease the supervised task.

High dimensional output space may be verbose and is likely to contain a lot of internal dependencies, especially if the task is not well understood or badly formulated. Starting from this observation, we propose to retain the DNN input pre-training (or regularization) idea and adapt it to the output space. This should provide a better target representation, leading to an easier learning process.

The remaining of this paper is as follows. In section 2, we describe the main principles of neural networks and the existing optimisation procedure for deep architectures. In section 3, we explain our double embedding proposition. Preliminary experiments on toy datasets are shown in section 4, as well as texture segmentation tasks and handwritten caracter reconstruction. Finally, the section 5 concludes.

2. Deep Networks

In the past decades, MLP have been studied to approximate non-linear functions in many decision tasks. The standard MLP relies on the original perceptron principle and its optimisation is based on the backpropagation algorithm. MLP are usually composed of two layers which perform the following operation:

$$\mathbf{h}_l(\mathbf{x}) = f_l(\,\mathbf{W}_l\,.\,\mathbf{h}_{l-1}(\mathbf{x}) + \mathbf{b}_l\,) \quad \forall l \in \{1, \ldots, N\}\ , \quad (1)$$

where $\mathbf{W}_l$ is a linear transformation, $\mathbf{b}_l$ is a vector of offsets, $\mathbf{h}_0(\mathbf{x}) = \mathbf{x}$, $\mathbf{h}_N(\mathbf{x}) = \hat{\mathbf{y}}$, and N is the number of layer in the complete network. f_l is a non-linear activation function. Deep architectures with more than two layers, suit for high dimensional problems, they are called Deep Neural Networks (DNN).

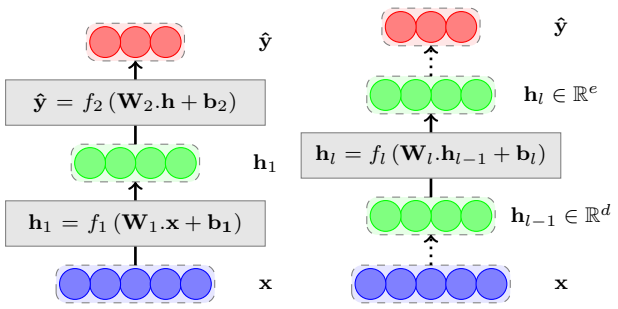


Figure 1. (left) a shallow MLP network, (right) a Deep Neural Network. For the layer l, d is the number of hidden units in $\mathbf{h}_{l-1}$ and e in $\mathbf{h}_l$; hence, $W_l \in \mathbb{R}^{d \times e}$, $\mathbf{b}_l \in \mathbb{R}^e$.

Two major propositions rose from the machine learning community to solve the problem of DNN. The first one stacks pre-trained auto-encoders, while the second one optimizes a regularized criterion. These two network building procedures learn embedding functions of the inputs in an unsupervised way in order to regularize the parameters of the network. An input embedding projects the input data in a better vector space where the neighborhoods are preserved. These unsupervised strategies are expected to be efficient as soon as an estimation of $p(\mathbf{x})$ is useful to learn the final supervised task: estimating $p(\mathbf{y}|\mathbf{x})$.

2.1. Deep layers pre-training and stacking

As deepest layers are hard to learn with a single backpropagation algorithm, [7, 1] propose to pre-train the parameters of these layers on an unsupervised learning with auto-encoder networks. The first layers of auto-encoder are stacked iteratively to build the complete DNN. Finally a supervised fine tuning optimizes all the parameters in the whole stacked network for the final task.

The auto-encoder architecture links a list of visible units ($\mathbf{v}$, the data), a list of hidden units $\mathbf{h}$ and a list of estimated visible units ($\hat{\mathbf{v}}$, the data reconstruction), as in Fig. (2). By doing so, the auto-encoder finds a reduced representation of the data in the hidden units. The auto-encoders could be either a RBM [7] or a derived form of the MLP network that is an Auto-Associator neural network (AA) [1].

Learning an Auto-Associator is similar to any MLP learning, using backpropagation. When using AA with C_{se} criterion and no non-linear function, we obtain a PCA [5]. However, the data compression induced by the C_{se} criterion might not be appropriate, when learning a binomial distribution. [8] propose to use the Kullback-Leibler divergence, also known as the cross entropy C_{ce}, as a reconstruction criterion,

$$C_{ce}(\mathbf{v}, \hat{\mathbf{v}}) = -\sum_k (v\ log(\hat{v}) + (1-v) log(1-\hat{v}))\ . \quad (2)$$

The unknown and evaluated distributions v and $\hat{v}$ characterize respectively $\mathbf{v}$ and $\hat{\mathbf{v}}$, then the Kullback divergence $KL(v, \hat{v})$ will be minimized when $v = \hat{v}$. This means when the reconstruction error of $\hat{\mathbf{v}}$ against $\mathbf{v}$ is minimum. To avoid trivial projection such as the identity function when the number of hidden unit is higher than the number of visible unit, [8] proposed to constrain the linear transformation matrices of the two functions $f(\mathbf{U}\mathbf{x}+\mathbf{c})$ and $g(\mathbf{V}\mathbf{h}+\mathbf{d})$ by making $\mathbf{U} = \mathbf{V}^T$. Whatever the auto-encoder is, we have to split it to keep aside the embedding function $f(\mathbf{U}\mathbf{x}+\mathbf{c})$, and drop the re-projecting function $g(\mathbf{V}\mathbf{h}+\mathbf{d})$. The embedding function is used to project all the input data into the hidden space, hence the hidden units are used as inputs for a new auto-encoder training in a new layer.

During the training of a new projecting function at layer l, every parameters of the deeper layers are fixed, only the weights matrix $\mathbf{U}_l$ and $\mathbf{c}_l$ are modified by the backpropagation. We can add more and more layers, like in Fig. (2,

right), depending on the complexity of the input space. Finally, an output layer is plugged on the top of the last layer. At that time, a full backpropagation algorithm is used to fine-tune all the parameters. Now, to maximise the capacities of the network, all the deep parameters are released so the backpropagation optimisation can reach them.

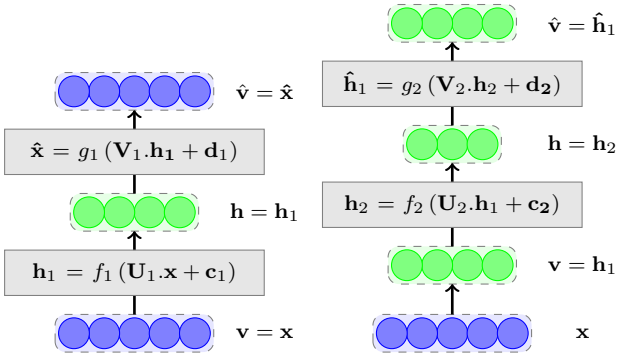


Figure 2. (left) An Auto-Associator learned on the input $\mathbf{x}$. (right) an Auto-Associator learned on the hidden units $\mathbf{h}_1$.

2.2. Regularized semi-supervised deep learning

The second proposition made by [14] learns deep neural networks via a double optimisation formulation. It consists in learning the two probability distributions $p(\mathbf{x})$ and $p(\mathbf{y}|\mathbf{x})$ simultaneously. Their approach optimizes a supervised criterion L_1 and the relevance of an embedding function $f(.)$ on unsupervised input data via a criterion L_2. The embedding is evaluated on its ability to reproduce into the projected space a similarity matrix $\mathbf{D}$ between the samples from the input space.

Let us define two sets of data: *$\mathcal{L}$, the Labeled observations*, and *$\mathcal{U}$, the Unlabeled observations*. Let $I_{\mathcal{L}}$ be the set of indexes of all elements belonging to the set $\mathcal{L}$, respectively $I_{\mathcal{U}}$ the set of indeces for the elements of $\mathcal{U}$. The multi-objective criterion is

$$\begin{aligned} \mathcal{J} = & \sum_{i \in I_{\mathcal{L}}} L_1\left(f(\mathbf{x}_i), \mathbf{y}_i\right) \\ + \lambda & \sum_{(i,j) \in (I_{\mathcal{L}} \cup I_{\mathbf{U}})^2} L_2\left(f_k(\mathbf{x}_i), f_k(\mathbf{x}_j), \mathbf{D}_k(i,j)\right) \quad . \end{aligned} \tag{3}$$

where λ is a balancing parameter to tune, and k is the layer at which the capacity of the embedding is evaluated.

At each iteration of the batch algorithm, they compute the stochastic gradient of this multi-objective function, until they reach a minima. This nice formulation of the embedding optimisation with an hinge loss as L_1 and a margin loss as L_2 (see [14]) provides competitive results to the state of the art methods as Deep Belief Network and DNN, on the MNIST dataset.

3. Embedding the Output Space

For high dimensional output tasks, we propose here to apply a similar embedding on the output space too. Some decision tasks, such as 2D-signal regression, or image segmentation, occurs in an high dimensional output space. This output space might be verbose and is likely to contain internal dependencies, especially if the task is badly formulated. Embedding on the output space might remove these dependencies and provide a better representation of the output. We propose here to learn embeddings for both the input data and the output labels. This allows us to optimize the decision function in a much smaller space which is adapted to the hidden complexity of the output. That is, an estimation of $p(\mathbf{x})$ and $p(\mathbf{y})$ will provide a good start to estimate $p(\mathbf{y}|\mathbf{x})$. We will describe both the stacked pre-trained encoders strategy and the multi-objective optimisation strategy. Nevertheless, described experiments are implemented through the stacking strategy.

Building the stacked network

We learn two embedding in an self supervised manner, one out of the input space, and a second one out of the output space. The input reduction is learned as in section 2.1, the AA are stacked iteratively. The projecting activation function f_l, the weights matrices $\mathbf{U}_l$ and the offset $\mathbf{b}_l$ are kept aside to initialize the $\mathbf{W}$ matrices of the final network (Fig. 2). The output reduction is done symetrically by an AA. Here, the output AA are identified with negative notations $-l$. The re-projecting function g_{-l}, the weights matrices $\mathbf{V}_{-l}$ and the offset $\mathbf{d}_{-l}$ are stored, and are stacked from the output units (Fig. 3). To sum up, the following inititialization of the complete network is done using :

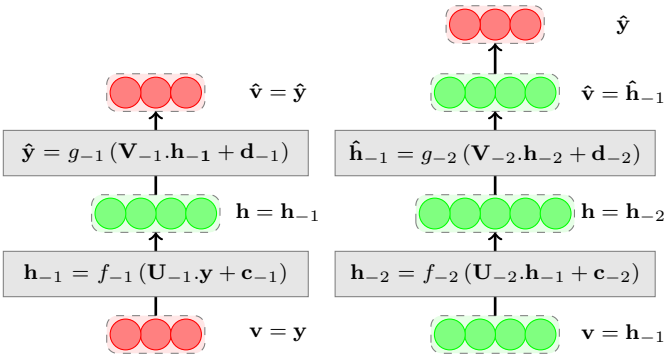


Figure 3. (left) An auto-Associator learned on the output $\mathbf{y}$ (right) An auto-Associator learned on the $\mathbf{h}_{-1}$ units.

$$\begin{array}{llll} \mathbf{W}_l \leftarrow \mathbf{U}_l; & \mathbf{b}_l \leftarrow \mathbf{c}_l; & l \in [1, T-1] & \\ \mathbf{W}_{N-l} \leftarrow \mathbf{V}_{-l}; & \mathbf{b}_{N-l} \leftarrow \mathbf{d}_{-l}; & l \in [0, N-T-1] & , \end{array} \tag{4}$$

where T is the layer where the projected input space is linked to the projected output space, and N is the total number of layer. For the final supervised tuning, the $\mathbf{W}_T$ matrix is randomly initialized, and all the $W_{l \neq T}$ matrix are released to permit backpropagation optimisation on the entire network .

Note that the regularised semi-supervised deep learning [14] can be extended through the addition of a third objective loss L_3 in equation 3 which evaluates the capacity of the output embedding.

4. Experiments

To demonstrate the validity of our proposition, we have run 4 types of experiments on 3 different datasets. We will first describe each dataset, then each type of experiment and finally we present the results of our experimentations. We generated three kinds of toy datasets for three different tasks: de-noising, simple phase recognition and texture recognition. All problems take as input an image and target the label of each pixel.

Task 1 Each example consists in a noisy image of a rectangle as an input, and the clean image of the rectangle as the target (64×64).

Task 2 Each example consists in a representation of the function $sin(i/8 + \phi) + cos(j/8 + \phi)$, as an input, where i and j are respectively the line number and the column number in the image, $\phi = 0$ on all the image except in a rectangle where $\phi = \pi/2$; the target is a white over black image of the rectangle (64×64).

Task 3 Each example consists in, as image input, a composition of two textures taken from the Brodatz texture archive [1]; the background is taken from Texture 64, a rectangle is drawn upon it with Texture 65. The target is a white over black image of the rectangle (128×128).

In all the datasets, input values are in between 1 and -1 and exactly 1 or -1 for the target. For each task, a train set of 1000 examples and a validation set of 100 examples have been generated. The first example of each validation set are shown on Table (1). A black and white rectangle image is really verbose and a lot of internal dependency occurs. Therefore a better representation of the output space should be available and pre-training on output should improve the results of the supervised task.

Experimental setup.

All the experiments have been run thanks to the Torch5 library [4]. We have built a DNN with two hidden layers. Each hidden layer consists in twice as much units as the size of one dimension of the input (64×2 pixels for the first dataset). Four experiments have been tested, each differs in the way the network is pre-trained:

[1] `http://www.ux.uis.no/~tranden/brodatz.html`

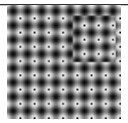
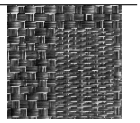
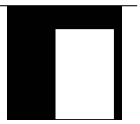
Task	1	2	3
Input			
Target			

Table 1. Image input and image target for the first validation example of the 3 tasks.

Task	Exp. 1	Exp. 2	Exp. 3	Exp. 4
1	0.154	0.153	0.068	**0.065**
2	0.133	0.132	0.051	**0.050**
3	0.167	0.167	**0.045**	**0.045**

Table 2. MSE at iteration 300 of the final supervised training on the validation set for the 3 tasks, for the 4 experiments.

Exp. 1 No pre-training, supervised learning by backpropagation is directly done.

Exp. 2 The first hidden layer is pre-trained by using an auto-associator on the input data. Standard supervised backpropagation is then used.

Exp. 3 The first and the second hidden layer are pre-trained by using, respectively, an auto-associator on the input data and an auto-associator on the output data. Standard supervised backpropagation is then used.

Exp. 4 The first and second hidden layers are pre-trained by using respectively an auto-associator on the input data and on the output data. The supervised learning is decomposed into two steps: on the first step, the link between the first and the second hidden layer is learned by backpropagation using the hidden state of the second auto-associator as target. On the second step, a standard supervised backpropagation is then performed on the whole DNN.

The pre-training for each auto-associator is done by batch gradient stopped at 50 iterations. The first step of the two supervised learning steps is also done by batch gradient with an early stopping at 50 iterations. Batch gradient with early stopping at 300 iterations is performed for the final supervised learning in all case. The learning rate for all the experiments is the same and has been fixed according to the first experiment (No pre-training). The four experiments have been undertaken on the 3 tasks.

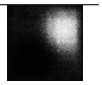
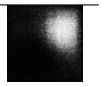
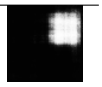
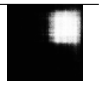
Iteration	10	100	200	300
Exp. 1				
Exp. 2				
Exp. 3				
Exp. 4				

Table 3. Output image evolution for the first validation example according to the number of iterations.

4.1 Results and discussions

Table (2) shows that, the validation error of the DNN is not improved significantly by input pre-training (Exp. 2), in comparison to standard learning (Exp. 1). However, it shows that with output pre-training (Exp. 3 and 4), DNN achieves better performance. Table (3) clearly shows that the output is already gathered at iteration 10 in a rectangle on Exp. 3 and 4, this is due to the fact that the DNN already knows about internal dependencies on the target thanks to pre-training. On Exp. 1 and 2, output looks more like a cloud at iteration 10. We need to wait until iteration 200 to start to see a rectangle.

According to the MSE at iteration 300, the 2-step supervised training in Exp. 4 does not seem to improve significantly the results in comparison to 1-step supervised training in Exp. 3. Nevertheless, the drop of the validation error is faster. Actually most supervised learning has been done during the first step. One may argue that pre-training is time consuming, so we may compare the result for fewer iterations on Exp. 3 and 4 than for Exp. 1 and 2; but even with less iterations, output pre-training does achieve better performances.

4.2 Comparison to KDE, the image reconstruction problem

After these experimentations based on toy datasets, we have compared our proposition to Kernel Dependency Estimation [13] on the image reconstruction problem. This relies on a kernel based loss.

A kernel function, $K(.,.) : \mathbb{R}^{d\times 2} \rightarrow \mathbb{R}$, can be used as a loss function that qualifies the efficiency of a supervised task. Effectively, a distance in an induced Hilbert space can be formulated thanks to its associated kernel $K(.,.)$,

$$L(\mathbf{y}, \hat{\mathbf{y}}) = K(\mathbf{y}, \mathbf{y}) - 2 * K(\mathbf{y}, \hat{\mathbf{y}}) + K(\hat{\mathbf{y}}, \hat{\mathbf{y}}) \ . \quad (5)$$

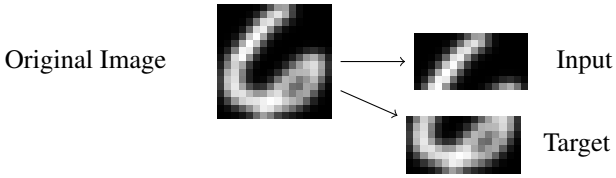


Figure 4. Decomposition of a USPS character in input image and target image.

For example, the RBF-loss, L_{RBF}, based on the RBF kernel, is expressed as

$$L_{RBF}(\mathbf{y}, \hat{\mathbf{y}}) = 2 - 2 * \exp\left(\frac{-||\mathbf{y} - \hat{\mathbf{y}}||^2}{2\sigma^2}\right) \ . \quad (6)$$

Highlighting this properties, [13] proposed to build two kernels, one for the input space and one for the output space. The authors then learn the dependency between these two kernels to achieve a supervised task. This is similar to our methodology, first of all, learn the unconditional probabilities of inputs and targets, $p(\mathbf{x})$ and $p(\mathbf{y})$ then try to learn the conditional probability of the target $p(\mathbf{y}|\mathbf{x})$. We reproduced the same image reconstruction task on USPS dataset presented in [13]. Each character image of 16×16 pixels is split into two smaller images, one corresponding to the top 8 lines and the other one corresponding to the bottom 8 lines. Fig. (4) shows this decomposition. Knowing the top lines, the inferning algorithm has to reconstruct the bottom lines. The first thousand examples of the database are taken into account in a 5-fold cross validation scheme. In an unusual manner, each training set consists in one fold (200 examples) and the corresponding testing set consists in the remaining 4 folds (800 examples). The RBF-loss, L_{RBF}, as defined in equation (6) with $\sigma = 0.35$, qualifies all the applied methods, KDE, k-NN, Hopfield-net and DNN. The results of the 3 first methods are reported from [13].

The DNN architecture has been fixed to 2 hidden spaces, $\mathbf{h}_1$ and $\mathbf{h}_{-1}$, that is 3 computational layers. Both hidden spaces belongs to $\mathbb{R}^{16}$. The learning rate and the number of iterations have been fixed according to the results of backprogation without pre-training. On pre-trained DNN, the first computational layer, from $\mathbf{x}$ to $\mathbf{h}_1$, is initialized thanks to an Auto-Associator on the input space (top 8 lines); the last computational layer, from $\mathbf{h}_{-1}$ to $\mathbf{y}$, is initialized thanks to an Auto-Associator on the output space (bottom 8 lines). Table (4) displays original and reconstruction characters produced by a DNN learned with pre-training.

The results presented in table (5) clearly shows the efficiency of Deep Architecture on this particular image reconstruction tasks. On the one hand, for the original experimentation scheme, DNN outperforms KDE, k-NN and Hopfield net. On the other hand, the L_{RBF} standard deviation

Character	Ex. 1	Ex. 30	Ex. 95	Ex. 101
Original				
Reconstruction				

Table 4. Characters reconstruction by DNN.

Methods	# folds	Pre-train	L_{RBF}
Hopfield net*	5	-	1.2190 ± 0.0072
k-NN *	5	-	0.8960 ± 0.0052
KDE *	5	-	0.8384 ± 0.0077
DNN	5	N	$\mathbf{0.8000} \pm 0.0167$
DNN	5	Y	0.8070 ± 0.0154
DNN	10	N	0.9489 ± 0.0258
DNN	10	Y	$\mathbf{0.9338} \pm 0.0225$

Table 5. RBF-loss of DNN, with or without pre-training, compared to KDE, k-NN and Hopfield Net (* results reported from [13]).

is greater when using DNN. Indeed, neural networks are prone to instability induced by random initialization. Pre-trained DNN leads to less deviation. Nevertheless their performance are similar to the standard backpropagation without pre-training. This observation can be explained by the fact that DNN without pre-training already performs well on this task and may have reached the limit of the given architecture. To emphasize the benefit of the pre-training, not only on the stability of the performance but also on the performance itself, we reproduced the experience but this time on a 10-fold scheme (100 examples in training and 900 examples in testing), making the task harder to achieve. Here, pre-training improves the RBF-loss L_{RBF} and its standard deviation compared to standard DNN.

5. Conclusion

In this study, a new approach for high dimensional problems such as image segmentation has been presented. The main contribution is to embed the inputs and the outputs of a given problem through a deep architecture in order to benefit from their internal dependencies.

A standard DNN can achieve good performances but seems to slowly converge. It may also fall into unwanted local minima. Input pre-training, by making a better representation of the input space, ease the supervised learning [7, 14]. Extending this idea, our proposition is based on output pre-training in order to also model output relations. This method achieves promising results, not only on toy-data image segmentation, but also on real image reconstruction task.

When compared with probabilistic methods (2D-HMM, 2D-CRF), the decision process of the proposed approach is really fast, but is limited to fixed input and output size problems. For instance, probabilistic models better suit variable-length signals such as human voice, handwriting, etc. Our future work will try to overcome this issue without falling into the pre-image curse.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[2] D. M. Bikel, R. L. Schwartz, and R. Weischedel. An algorithm that learns whats in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[3] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. *ECCV*, 2008.

[4] R. Collobert, S. Bengio, L. Bottou, J. Weston, and I. Melvin. Torch 5, 2008. `http://mloss.org/software/view/128/`.

[5] K. Diamantaras and S. Kung. *Principal component neural networks Theory and applications*. John Wiley, New York,, 1996. isbn 0-471-05436-4.

[6] A. El-Yacoubi, M. Gilloux, and J.-M. Bertille. A statistical approach for phrase location and recognition within a text line: An application to street name recognition. *IEEE Trans. on PAMI*, 24(2):172–188, 2002.

[7] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comp.*, 18(7):1527–1554, July 2006.

[8] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *JMLR*, 10:1–40, 2009.

[9] S. Nicolas, T. Paquet, and L. Heutte. A markovian approach for handwritten document segmentation. *ICPR*, 3:292–295, 2006.

[10] L. R. Rabiner. A tutorial on hidden markov models and selected apllications in speech recognition. In *Readings in Speech Recognition*, pages 267–296. Kaufmann, 1990.

[11] M. Szummer and Y. Qi. Contextual recognition of hand-drawn diagrams with conditional random fields. *IWFHR*, pages 32–37, 2004.

[12] G. Tsechpenakis, J. Wang, B. Mayer, and D. Metaxas. Coupling CRFs and deformable models for 3D medical image segmentation. *ICCV*, Issue 14-21:1 – 8, 2007.

[13] J. Weston, O. Chapelle, A. Elisseeff, B. Scholkopf, and V. Vapnik. Kernel dependency estimation. *NIPS*, 2002.

[14] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. *ICML*, pages 1168–1175, 2008.