Abstract—In this paper we propose a modification of the Cognitive Architectures for Sensory Processing proposed by Chalasani and Principe. Here we keep the bottom-up data representation through generative models as before, but propose a top-down flow based on backpropagation of gradients for recognition. By treating the bottom-up procedure involved in the inference step as a recursive neural network, we show that supervised learning can be used in conjunction with other layers commonly used for Deep Learning. Also, this allows us to learn models that incorporate at the same time data classification and statistical modeling of the input. We show that this combination provides classification results that are robust to input noise.

I. INTRODUCTION

Understanding the dynamical interactions of bottom-up sensory input perception and top-down environment interpretation is fundamental for studying the cognitive power of the human nervous system [1]. On the other hand, an algorithm that enhances its signal processing quality (bottom-up) through modeling and interpretation (top-down) of its environment can be considered a giant step for real world scene analysis. For investigating that problem, Principe and Chalasani recently proposed a hierarchical distributed Bayesian formulation of a cognitive architecture [2], dubbed Deep Predictive Coding Networks (DPCN). That architecture was trained for object recognition in video. Intuitively, DPCNs models videos as stochastic processes where higher order statistics are Markovian. The power of the method relies on the ability of the architecture to learn through its hierarchical representations which statistics should be considered important and evolving through time with the Markov assumption. This way, the top layers influence the bottom layers as nonlinear causes. The bottom layers feed the top ones with sensory information filtered using the Markov assumption provided by the top ones. They showed that this bidirectional flow can be used by the architecture to organize itself with features that resemble the early visual cortex [2].

Although self-organization and unsupervised learning are crucial for developing brain-like sensory processes [3], from a practical perspective, supervised learning provides better results at tasks such as classification [4]. This is what led to the resurgence of interest in Neural Networks, rebranded as Deep Learning. One of the main aspects that allowed those networks to obtain good results can be credited to advanced knowledge about the theory of learning, faster computer, and efficient ways of handling large data sets such as unsupervised pre-training [5] and more recently Dropout [6].

Unsupervised pre-training, e.g. Restricted Boltzmann Machines [7] and Auto-encoders [8], can be considered effective initialization techniques because they place the parameters of the network at suitable positions, with bounded magnitudes, that speed up the convergence of supervised algorithms that follow them. Dropout [6] is based on randomly dropping out hidden units during the training phase and renormalization in the test phase. This technique has proven to be a practical and efficient approximation to neural network ensemble averaging. When used together with the \(L_2\)-norm constraint or weight decay, Dropout trained networks have sparse activations and higher accuracy than networks trained with \(L_1\)-norm constraints [9].

More powerful Deep Neural Networks (DNN) were recently trained by using simple new types of layers, such as Rectified Linear (ReLU) [4] and Maxout [10]. ReLU is a piecewise approximation to soft plus nonlinearity, i.e. the limit of a specific series of sigmoid units [4]. Maxout units [10] are based on parallel linear inner product operators that share the same input but where only the maximum resulting product is feed forward to the next layer.

All these layer modifications can be seen as analytical methods, where a deterministic noninvertible operation transforms the input of each layer. This increases the abstraction about the inputs and favors better discrimination. But, on the other hand, these same layers cannot be used, for instance, for filtering and missing input filling once they are trained for classification. With the intent to create a method that learns probabilistic models of the data, while training for pattern recognition, we propose to employ Deep Predictive Coding Networks (DPCN) recently developed in our laboratory and augment them with targets for classification.

We propose to use target information as the top layer cause in DPCN, which brings an external goal to the generative modeling naturally produced by empirical Bayes. Since the top layer cause affects the training of the top layer state model, this effect ripples through inference of all the bottom layers as well as on the learned parameters. This is a rather complex process that can be understood by the backpropagation framework as we will demonstrate.

As a proof of concept of the proposed idea, we investigate a static classification problem, which can be interpreted as sparse coding as preprocessing for image recognition. In this application, the state model in DPCN will be substituted by a linear model that will relate the input of the layer to the target that is propagated from the top cause. Note that
this differs from the previous works [11] [12] in the sense that we keep a reconstruction cost of the inputs, contrary to [13] that simply regularizes the output activations of the layers. Also, our method differs from [14], which also uses input reconstruction costs, since here we do not constrain the encoding coefficients to be orthonormal. We choose to feed forward the hidden factors of the model themselves or reconstructions of the inputs provided by the generative bottom-up flow. Note that propagating a reconstruction of the input is a filtering process. In a previous work, we showed that adapting the filtering and/or preprocessing together with the whole network provided better results at EEG classification benchmarks [15].

An important question that the knowledgeable reader may ask at this point is, “How can the gradients be backpropagated, since those layers receive the activations of the layer below as input, but forwards out a model of that input?”.

To answer this question here we note that the generative inference procedure is a function of the inputs and thus, can be seen as a recursive neural network where the gradients can be backpropagated in the next optimization step.

The reminder of this paper is organized as follows. In Section II we propose the functions of our model and comment about the details of the bidirectional flow. In Section III we present classification and filtering results using neural networks trained with our model and compare to state-of-the-art architectures used in Deep Learning. We address a brief discussion in Section IV. Finally, we conclude the paper.

II. PROPOSED ARCHITECTURE

In accordance to the framework proposed in [2], we have to define an energy function and how it is affected by the bottom-up and another for the top-down of information flow. Below is what we proposed in this work.

Bottom-up: Let us assume that $A$ is a set of filters and $s$ a set of sparse coefficients. We have the Sparse Coding model [3] when the energy function of the probabilistic formulation of our model is given by

$$E \propto (x - As)^2 + \lambda |s|_{L1},$$  \hspace{1cm} (1)

where $x$ is the input data. Just like in [2], we can use an expectation-maximization (EM) algorithm optimizing (1) for $s$ and $A$ alternately, using a smooth approximation for the $L_1$ norm. After one optimization step we forward either the sparse code $s$ or the input reconstruction $As$ to a multi-layer neural network for recognition. Here, the studied neural network has one hidden layer with a ReLU activation function (i.e. $f(x) = \max(x, 0)$) and the output layer has a Softmax nonlinearity with negative log-likelihood as cost function for classification, see Fig. 1. Note that we could use other types of layers instead of the ReLU, but the investigation of the other possibilities is beyond the scope of this paper.

Top-down: The backpropagation of gradients from the negative log-likelihood is what we propose to augment the top-down information flow in [2]. Note that here, since this task is static, we cannot rely on the same temporal memory. Nevertheless, we note that labels are assumed available, i.e. provided by a content addressable-memory whose values were consolidated through the action-perception-reward loop. This way, the parameters $A$ are self-organizing to represent the input data taking into consideration both the data statistics and the context, encoded by its labels. In order to derive the gradients through the generative model part of the network, let us assume a simple stochastic gradient descent optimization of (1). Thus, we adapt the code $s$ using

$$s \leftarrow s + \eta (x - As)A^T + f'(s),$$  \hspace{1cm} (2)

where $f(s)$ is a smooth approximation to the absolute function and $f'(s)$ is its derivative, for example $f(s) = \sqrt{s^2 + \epsilon}$, where $\epsilon$ is a small positive constant. Assuming that we repeat (2) for $N$ steps (or until convergence) that cost function depends on all $N$ evaluations of (2). Successive application of the chain rule gives us the equation for the backpropagation through optimization step:

$$\delta_n = 1 + \eta A^T + f''(s) \delta_{n-1},$$  \hspace{1cm} (3)

where the total gradient is the result of a summation that starts at $n = N$ and goes down to $n = 1$, just like backpropagation through time for recurrent neural networks [16].

It is important to note that other optimization procedures could also be considered, for instance, BFGSs, Conjugate Gradient, FISTA [2], etc., all define recursive and smooth functions where the gradients can backpropagated through. Here, for instance, we use RMSprop [17] to optimize $s$, since it normalizes the gradients using an exponential window and helps to deal with “narrow valleys” [17] that are common when dealing with highly correlated inputs, such as the case of sparse coding of images.

A schematic block diagram of the proposed architecture is shown in Fig 1.

III. EXPERIMENTS

In this section we implement experiments with the MNIST dataset for digit recognition. We compare 3 models. The first (Model 1) has the architecture described here but only propagates the coefficients $s$ at the first layer as in sparse coding. The second (Model 2) has the same architecture but the first layer propagates the filtered inputs given by $As$ as proposed. (Model 3) is composed by 2 Rectified Linear layers followed by a Softmax layer. All the models have the same number of dimensions as in [6]: 1200 neurons for the first two layers and 10 output neurons, one for each class. All the models are trained using Stochastic Gradient Descent with Dropout. We use early-stopping with a patience of 100 epochs to determine convergence. Again, the code for reproducing our results is based on Pylearn2, thus it is highly extensible for other architectures and easy experimentation. Please, refer to the code for further parameters.

In Table I we show accuracy of the compared networks for several levels of Gaussian noise corruption in the input data. We repeated the experiments 10 times randomizing the
Gaussian corruption. We calculated the mean and standard deviation of the number of misclassified examples for each network. Also, we applied a t-test ($\alpha = 0.05$) to confirm that differences in mean are statistically significant. As expected, our proposal provides competitive results compared to the state-of-the-art, but since the first layer multi-tasks between modeling the input data and projecting it to a separable space, we obtained slightly lower accuracy for noiseless input. On the other hand, for higher noise levels, the generative layers that we proposed obtained better results. Also, we observed that the generative layers succeeded at filtering the inputs while learning the classifier which is the original goal of our approach. To further observe this phenomenon, we show in Fig. 2 sample noisy inputs and the respective reconstructed inputs outputted by the first layer of Model 2.

### TABLE I

| Classification accuracy (number of errors; total is $10^4$) as a function of Gaussian noise std. ($\sigma$) of the compared methods at MNIST benchmark |
|---|---|---|
| $\sigma = 0$ | 480 | 254 | 126 |
| $\sigma = 0.5$ | $1668 \pm 28$ | $664 \pm 18$ | 759 $\pm 18$ |
| $\sigma = 1$ | $4398 \pm 42$ | $2649 \pm 31$ | $3523 \pm 33$ |

*a* First layer is generative and outputs sparse code.

*b* First layer is generative and outputs reconstructed data.

*c* First layer regular feedforward network. See text for details.

Note that although Model 2 has the same number of neurons as the other methods, only its last hidden layer is directly working for transforming the data to a space that is linearly separable. The first layer is focused in filtering the input in a way that makes the aforementioned transformation easier. Thus, this generative model recreates versions of the input that are easier to classify. This technique using fewer neurons than the data dimension would be highly promising for task-driven data compression [18]. We plan to investigate this in future work.

### IV. DISCUSSIONS

In the experimental section we validated (for linear sparse coding models) the concept of training Generative Models with backpropagated errors from external targets. As we can observe in Table I the model that outputs the input data reconstruction achieves better results than does the model that output the sparse codes (model 1). The approach of outputting the code instead of a model of the data, is the
one that is mostly used in DNN research [12].

Nevertheless, all the results above can be improved with longer runs of the optimization step. In the present experiments, we run the expectation step (optimizing for $s$) only 10 times for speed purposes (all the hyperparameters and implementation details can be seen at the companion source code). This paper is proof-of-concept, but precise cross-validation of this parameter is still needed. Another way to improve the results is using better Generative Models; for instance, our future investigation will focus on convolutional [19] models.

Beyond that, we believe that the even more interesting results should be obtained for problems where the input data is time varying as in video processing. This would allow us to use state models as in DPCNs augmenting the reconstruction is time varying as in video processing. This would allow us to improve the results is using better Generative Models; for instance, our future investigation will focus on convolutional [19] models.

V. CONCLUSIONS

In this paper we proposed a strategy for the joint optimization of Generative Models and classification networks. Here, the generative model is treated as a layer of the network. This way we showed that the backpropagation algorithm synergizes with expectation-maximization used to fit the model. Generally, the expectation step was focused on minimizing the error between the reconstructed and original data while the maximization multi-tasked between that same error minimization and improving classification results. In order to do that, we treated the optimization procedure as a recursive neural network and used gradient derivatives similar to the in backpropagation through time.

We showed that using our method, we could train a network where the generative model outputs either its hidden factor or a filtered version of the data. We observed that the model that outputs the filtered data obtained better results than did a regular DNN when a high level of noise is present in the input.

REFERENCES