Multivariate Regression with Incremental Learning of Gaussian Mixture Models

Juan M. ACEVEDO-VALLE [a,1], Karla TREJO [a], Cecilio ANGULO [a]
[a] *Knowledge Engineering Research Group, Department of Automatic Control, Universitat Politècnica de Catalunya - BarcelonaTech, Spain*

Abstract. Within the machine learning framework, incremental learning of multivariate spaces is of special interest for on-line applications. In this work, the regression problem for multivariate systems is solved by implementing an efficient probabilistic incremental algorithm. It allows learning high-dimensional redundant non-linear maps by the cumulative acquisition of data from input-output systems. The proposed model is aimed at solving prediction and inference problems. The implementation introduced in this work allows learning from data batches without the need of keeping them in memory afterwards. The learning architecture is built using Incremental Gaussian Mixture Models. The *Expectation-Maximization* algorithm and general geometric properties of Gaussian distributions are used to train the models. Our current implementation can produce accurate results fitting models in real multivariate systems. Results are shown from testing the algorithm for both situations, one where the incremental learning is demonstrated and the second where the performance to solve the regression problem is evaluated on a toy example.

Keywords. regression problem, incremental learning, gaussian mixture models

1. Introduction

Gaussian Mixture Models (GMMs) are appealed frequently in machine learning applications and related areas, since they are among the most used methods for clustering. In such a context, they are commonly employed in tasks where it is necessary to model complex and nonlinear parameters [1]. However, recently they have been actively applied to solve the regression problem and they have also been used to model high dimensional, non-linear redundant maps [2,3,4].

On the other hand, incremental learning algorithms might play a critical role in many applications. Those algorithms consider the learning scenario for streaming data arriving over time and have been widely applied in machine learning, pattern recognition, data mining, and fuzzy logic [5,1,6]. In [6], a summary of the challenges for incremental learning is presented, furthermore some of the main techniques that have been applied to solve the problem are described. In general, most of the machine learning techniques have been extended to cover the incremental learning paradigm opening the door to new

[1]Corresponding Author: Automatic Control Department, Universitat Politècnica de Catalunya, Pau Gargallo 14, 08028 Barcelona, Spain; E-mail: juan.manuel.acevedo.valle@upc.edu.

challenges (e.g. Support Vector Machines, Decision Trees, Genetic Algorithms, Gaussian Mixture Models, among others.). The reader is refereed to [6] for further details.

Incremental learning using GMMs has been previously studied with more emphasis on its applications as a semi-supervised classifier method and density distribution estimator [5,1,7]. However, in this work we focus on its suitability to solve the regression problem within the incremental learning framework.

The architecture presented in this work is suitable, but not limited, to solve the problem of modeling input-output multivariate systems. Data feeding the system is not available at the beginning, but it is collected incrementally in data batches of input-output data points. Therefore, the model is trained each time a new data batch is available and afterwards that data batch is discarded.

The main contribution of this paper is twofold: to combine two interesting mechanisms, incremental learning and regression, using GMM and providing the source code for those researchers who are interested in testing the approach in their own work[2]. This work uses simple examples to make it easier for the reader to understand the approach, but we have also tested the approach in more complicated systems as the vocal tract in [3,4] even though in those works the incremental learning mechanisms were different to the one presented here.

The reminder of this paper is organized as follows. Section 2 introduces the incremental learning algorithm and the regression mechanism for multivariate systems. Section 3 presents a simple example that shows how the incremental learning mechanism works. Section 4 introduces a simple non-linear redundant input-output system used to illustrates how the regression mechanism works to solve the inference problem. Finally, a brief conclusion is presented in Section 5.

2. Learning Algorithm

The learning procedure consists of two main mechanisms: the first one is the traditional *Expectation-Maximization* algorithm (EM-algorithm) training algorithm for Gaussian Mixture Models; the second step is the *incremental mechanism* that allows to include new knowledge in previously trained GMMs.

First of all, we assume a multivariate input-output system $\mathbf{x} = f(\mathbf{y})$, for which experiments are run to generate data batches of the extended vector $\mathbf{z} = [\mathbf{x}, \mathbf{y}]^T$. Then, a starting GMM is computed for the system described by $\mathbf{z} \in X \times Y$ using the initial data. A Gaussian Mixture Model is defined by the set of parameters $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{K}$, where π_j, μ_j and Σ_j are respectively the prior probability, the distribution mean and the covariance matrix of the j-th Gaussian, for $j = 1, 2, ..., K$, being K the number of Gaussian components. In Algorithm 1, the incremental learning algorithm used to train a GMM using data batches is summarized.

Algorithm 1 is fed with the following parameters: the minimum and maximum number of Gaussian components in the model, K_{min} and K_{max} respectively, the maximum number of Gaussian components that can be added to the model at each training step, ΔK_{max}, and the forgetting rate, α. In line 2, the GMM is initialized using the first batch of data, B_0, the *getBestGMM* function computes a GMM for each value of K within the allowed interval $[K_{min}, K_{max}]$. From those models, the one that best fits the data batch

[2]Code related with this work is available at https://github.com/yumilceh/igmm.

according to the *Bayes Information Criterion* (BIC), which is based on the maximum likelihood function, is selected. We will call the selected model, *gmm*. In *getBestGMM*, the GMMs are obtained using the EM-algorithm implemented in the open-source library *scikit-learn*[3], but also available in other open source tools (i.e. *TensorFlow*, *Open-CV*, and others).

Algorithm 1 Growing Gaussian Mixture Model Algorithm

```
1: Set parameters: $K_{min}$, $K_{max}$, $\Delta K_{max}$, $\alpha$.
2: $gmm \leftarrow getBestGMM(B_0, K_{min}, \Delta K_{max})$
3: for $B_i$ in $\mathbf{B}$ do
4:     $gmm_{new} \leftarrow getBestGMM(B_i, K_{min}, \Delta K_{max})$
5:     $gmm_{new}.gauss[:].w \leftarrow \alpha * gmm_{new}.gauss[:].w$
6:     $gmm.gauss[:].w \leftarrow (1-\alpha) * gmm.gauss[:].w$
7:     $\mathbf{S_{DKL}} \leftarrow getKLDiveregence(gmm, gmm_{new})$
8:     while $gmm.k + gmm_{new}.k > K_{max}$ do
9:         $i, j = argmin(\mathbf{S_{DKL}})$
10:        $gmm.gauss[i] = merge(gmm.gauss[i], gmm_{new}.gauss[j])$
11:        $delete(gmm_{new}.gauss[j])$,  $\mathbf{S_{DKL}}[i, j] = \infty$
12:    end while
13:    $gmm \leftarrow join(gmm, gmm_{new})$
14: end for
```

From line 3, the model is trained every time a new data batch B_i is available. In line 4, a new GMM, gmm_{new}, is computed feeding the *getBestGMM* function with B_i. In lines 5 and 6, the prior of each component in *gmm* and gmm_{new} is updated, respectively. The prior's update is done according to the forgetting rate, α.

The most important step for the incremental learning mechanism is the merging of Gaussian components. Choosing which components of gmm_{new} will be merged to which components of *gmm* is the most challenging task of our approach. Therefore, before any components could be merged, a divergence matrix is obtained to evaluate the similarity between Gaussian components in gmm_{new} and *gmm*. Similarly to [1], we consider the *Kullback-Leibler divergence* (KLD) defined as

$$DKL(g_1, g_2) = \log\left(\frac{|\Sigma_2|}{|\Sigma_1|}\right) + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_1^{-1} (\mu_2 - \mu_1) - D, \qquad (1)$$

but as it is not a symmetric measure, we use a symmetrized version defined as

$$s_{DKL} = \frac{1}{2}\left(DKL(g_1, g_2) + DKL(g_2, g_1)\right). \qquad (2)$$

Finally, lines 8-12 of Algorithm 1 represent the merging process. Therein, the most similar Gaussian component in gmm_{new} is merged to its most similar counterpat in *gmm* and dropped. This process is repeated until the sum of components between both models is not larger than the maximum number of components K_{max}. Based on the geometric properties of Gaussian functions, the merge operation is shown in in Algorithm 2 from

[3]http://scikit-learn.org/stable/

[1]. Finally, if after the meging process there are remaining components in gmm_{new}, then they are included in gmm in line 13.

Algorithm 2 Algorithm to merge two Gaussians

```
merge(gaus_1, gaus_2)
 1: $f_1 = \frac{w_1}{w_1+w_2}, \quad f_2 = \frac{w_2}{w_1+w_2}$
 2: $w = f_1 + f_2$
 3: $\mu = f_1\mu_1 + f_2\mu_2, \quad \Sigma = f_1\Sigma_2 + f_2\Sigma_2 + f_1 f_2(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$
 4: $gauss \leftarrow w, \mu, \Sigma$
 5: return gauss
```

Algorithm 3 Algorithm to infer complete data from incomplete data

```
infer(x, gmm, nn)
 1: $\mathbf{d} \leftarrow zeros(gmm.k, 1)$
 2: for $i$ in $range(gmm.k)$ do
 3:     $\mathbf{d}[i] \leftarrow |gmm.gauss[i].\mu - \mathbf{x}|$
 4: end for
 5: $\mathbf{idx} \leftarrow argsort(\mathbf{d})$ % ascending order
 6: $\mathbf{Y} = zeros(lenght(gmm.gauss[0].\mu) - lenght(\mathbf{x}), nn), \quad \mathbf{P} = zeros(nn, 1)$
 7: for $i$ in $range(nn)$ do
 8:     $\mu_i \leftarrow gmm.gauss[idx[i]].\mu, \quad \Sigma_i \leftarrow gmm.gauss[idx[i]].\Sigma$
 9:     $\mathbf{Y}[:, i] = \mu_i^y + \Sigma_i^{yx}(\Sigma_i^x)^{-1}(x - \mu_i^x)$
10:     $\hat{\mathbf{z}}_i = [\mathbf{x}, \mathbf{Y}[:, i]]^T$
11:     $\mathbf{P}[i] = \pi_i \frac{1}{\sqrt{(2\pi)^K|\Sigma_i|}} e^{-\frac{1}{2}\left((\hat{\mathbf{z}}_i - \mu_i)^T \Sigma_i^{-1}(\hat{z}_i - \mu_i)\right)}$
12: end for
13: return $\mathbf{Y}[:, argmin(\mathbf{P})]$
```

2.1. Regression method

This regression mechanism follows our previous works [3,4] and it is based on Gaussian Mixture Regression (GMR) from [8]. It is summarized in Algorithm 3. An n-dimensional input space $X \in \mathbb{R}^n$ is mapped onto an m-dimensional output space $Y \in \mathbb{R}^m$. Thus, the function $\mathbf{y} = f(\mathbf{x}) + \varepsilon$ is assumed, where $\mathbf{y} \in Y$, $\mathbf{x} \in X$ and ε is random noise. A GMM is trained using the algorithm introduced before with data batches of couples $(\mathbf{x}, \mathbf{y})$.

To obtain the input $\mathbf{x}$ that maximizes the probability to produce the output $\mathbf{y}$, the GMR process first defines the partitioned vector $\mathbf{z} \in X \times Y$, where

$$\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}. \tag{3}$$

For each Gaussian j in the GMM the partitions

$$\mu_j = \begin{pmatrix} \mu_j^x \\ \mu_j^y \end{pmatrix} \quad \text{and} \quad \Sigma_j = \begin{pmatrix} \Sigma_j^x & \Sigma_j^{xy} \\ \Sigma^{yx} & \Sigma_j^y \end{pmatrix}, \tag{4}$$

are considered to compute the conditional probability distribution $P_j(X \mid \mathbf{y}) \sim N_j(\hat{\mu}_j, \hat{\Sigma}_j)$ over the input space X given a desired output $\mathbf{y}$, where

$$\hat{\mu}_j = \mu_j^y + \Sigma_j^{yx}(\Sigma_j^x)^{-1}(x - \mu_j^x) \quad , \quad \hat{\Sigma}_j = \Sigma_j^y + \Sigma_j^{yx}(\Sigma_j^x)^{-1}\Sigma_j^{xy}. \tag{5}$$

Considering that $P(X \mid \mathbf{y})$ is at its maximum when $\mathbf{x} = \hat{\mathbf{x}}_j = \hat{\mu}_j$, then a natural selection for $\mathbf{x}$ in order to produce $\mathbf{y}$ is $\hat{\mathbf{x}}_j$. But we have K candidates for $\mathbf{x}$, hence it is necessary to compute the probability of the vector $\hat{\mathbf{z}}_j = [\hat{\mathbf{x}}_j,\ \mathbf{y}]^T$ belonging to its generator Gaussian as

$$P(\hat{z}_j) = \pi_j \frac{1}{\sqrt{(2\pi)^K |\Sigma_j|}} e^{-\frac{1}{2}\left(\left(\hat{\mathbf{z}}_j - \mu_j\right)^T \Sigma_j^{-1} \left(\hat{\mathbf{z}}_j - \mu_j\right)\right)}, \tag{6}$$

and finally the point $\mathbf{z}^* = \hat{\mathbf{z}}_j$ that maximizes $P(\hat{\mathbf{z}}_j)$ is selected as the point that better fits the model. In other words, according to our prior knowledge of $f(\mathbf{x})$, $\mathbf{z}^* \in f(\mathbf{x})$, we infer that the output $\mathbf{y}$ is generated by $\hat{\mathbf{x}}_j$.

It is also interesting to mention that, in order to minimize computation time to obtain only $\mathbf{x}$, the regression might be restricted to the k-nearest Gaussian components to $\mathbf{y}$ considering their mean partition μ_j^i. As observed in Algorithm 3, our regression mechanism considers nn nearest neighbors. Finally, depending on how the partitions are defined in Eqs. (3)-(4) the mechanism can be used either for inferring $\mathbf{x}$ from $\mathbf{y}$, or for predicting $\mathbf{y}$ from $\mathbf{x}$.

3. Incremental Learning Example

In this section we present a simple example to illustrate the growth of a GMM using our incremental learning algorithm. We consider data batches randomly generated from 2-dimensional Gaussian distributions. Those data batches arrive at three different times. Figure 1 shows training steps at $t-2$ and $t-1$, whereas Figure 2 shows training step at t. The parameters considered to obtain that figure are $K_{min} = 2$, $K_{max} = 5$, $\Delta K_{max} = 5$ and $\alpha = 0.05$.

In Figure 1, it is observed that at time $t-2$, the model *gmm* obtained with the data batch is a mixture with two components. As it is the first step, the model is the result of the pure EM-algorithm choosing the number of components that maximizes the BIC as indicated in line 2 of Algorithm 1. Then, at time $t-1$ a new data batch is available as showed at the central plot in Figure 1. The GMM gmm_{new} that best fits the new data is selected as indicated in line 4 of Algorithm 1. The next step is to merge the Gaussian components of *gmm* and gmm_{new} until the total number of components is lower or equal to K_{max}. Notice in the figure that the only components which are merged are those colored in red.

A second step of incremental learning is observed in Figure 2 after a new data batch arrives. Again a GMM gmm_{new} is fitted to the new incoming data using the EM-algorithm and the BIC criteria. In this case as the number of components of the model has already reached its maximum ($K_{max} = 5$), all the components in gmm_{new} must be merged to the components in *gmm*. Figure 2 indicates with distinctive colors which components in gmm_{new} and *gmm* are merged to obtain the new *gmm*.

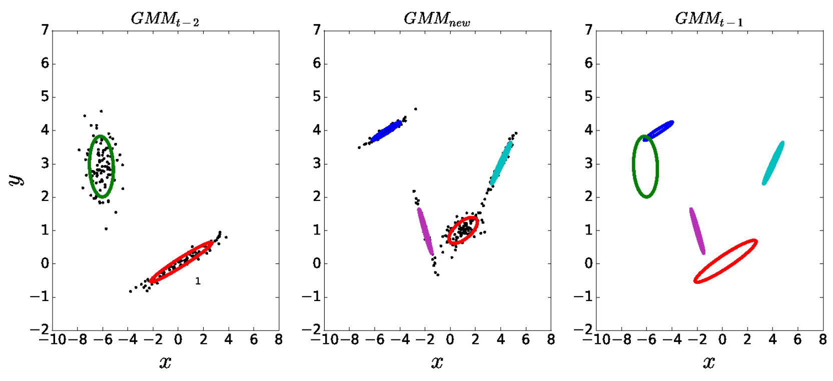

Figure 1. Incremental learning of a GMM. At $t-2$ pure EM-algorithm is used to initialize the model at $t-1$. The proposed incremental learning mechanism is used at time $t-1$ to growth the model with new incoming data.

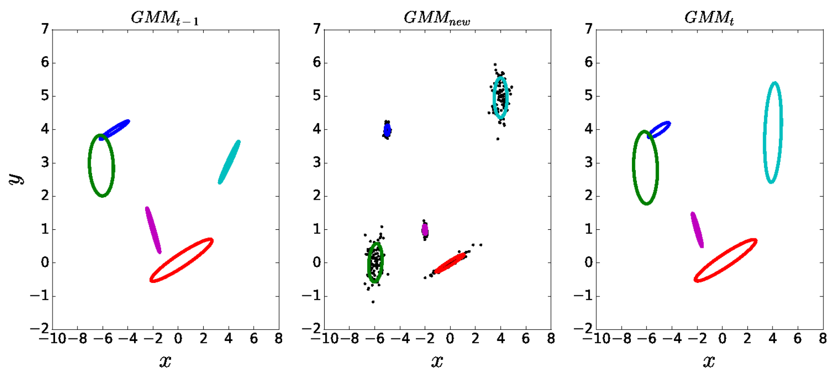

Figure 2. Incremental learning of a GMM. A model that has been already initialized is trained with our incremental learning mechanism at time t with new incoming data.

Figure 2 contains relevant information to understand what happens at the incremental learning level mechanism. Consider for instance the green component, the mechanism chooses to mix these two components based on the s_{KLD} measure, however the result in gmm_t is very similar to the component in gmm_{t-1}. This is attributed to the prior weight w of each Gaussian, the prior weights in gmm_w are scaled by the forgetting factor. Thus, when f_1 and f_2 are computed in line 1 of Algorithm 2, the component that is already in the model is considered more relevant and it is slightly modified by the new data. It means that it is very important to choose a good value for the forgetting factor α which not necessarily must be constant. An adequate value for α will depend on the system to be modelled, the size of the incoming data batches and the mechanisms used to draw those data batches. For example, in former works [3,4], we adopt an active learning architecture inspired on [9] and [2] to draw data batches in order to maximize a measure of learning rate. In general, choosing K_{min}, K_{max} and ΔK_{max} will depend on the complexity of the system to be modelled.

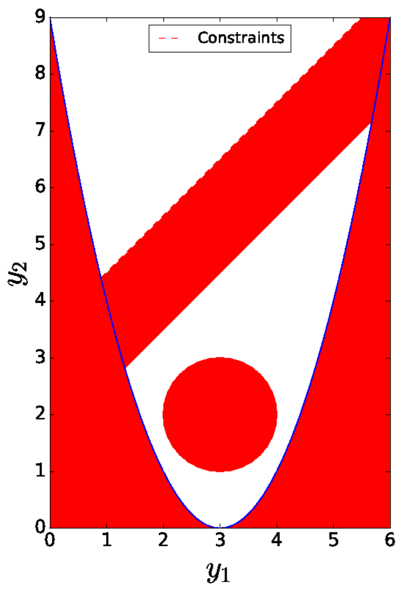

Figure 3. Parabolic Shaped Region System

4. Regression Problem Example

In order to show the performance of our modeling approach we proposed the inference problem $\mathbf{x} = f(\mathbf{y})$ for the simple toy example represented by

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f(\mathbf{x}) = \begin{pmatrix} x_1 \\ (x_2 - 3)^2 \end{pmatrix} \text{ and } c = \begin{cases} 1 \text{ if } \mathbf{y} \in \text{constraints} \\ 0 \text{ elsewere} \end{cases} \tag{7}$$

where y_i are the components of the output space, x_i the components of the input space and c is a signal indicating if constraints are violated or not. As it is observed in Figure 3, the output-space projection is a parabolic shaped region where the red regions represent constraints.

When constraints are violated then $\mathbf{y}$ takes the closest value $(y_1, y_2)^T$ on the valid region. Both input components are constrained to the interval $[0,6]$, whilst output dimensions are constrained to the white region and its blue borders in Figure 3. Therefore, due to the definition of the system it is non-linear, constrained and redundant which makes this system interesting to study in a simple-fashion manner the validity of our approach before applying it to more complex systems with a higher number of dimensions.

In order to show some examples with the regression mechanism for the parabolic shaped region system, we propose to train three 4-dimensional GMM using the incremental learning mechanisms from Algorithm 1. Each model is trained with random data obtained using different random seeds.

First of all, the parameter chosen to generate the models are: $K_{min} = 3$, $K_{max} = 10$, $\Delta K_{max} = 7$ and $\alpha = 0.2$. The model is trained with batches of data with 15 couples (x, y),

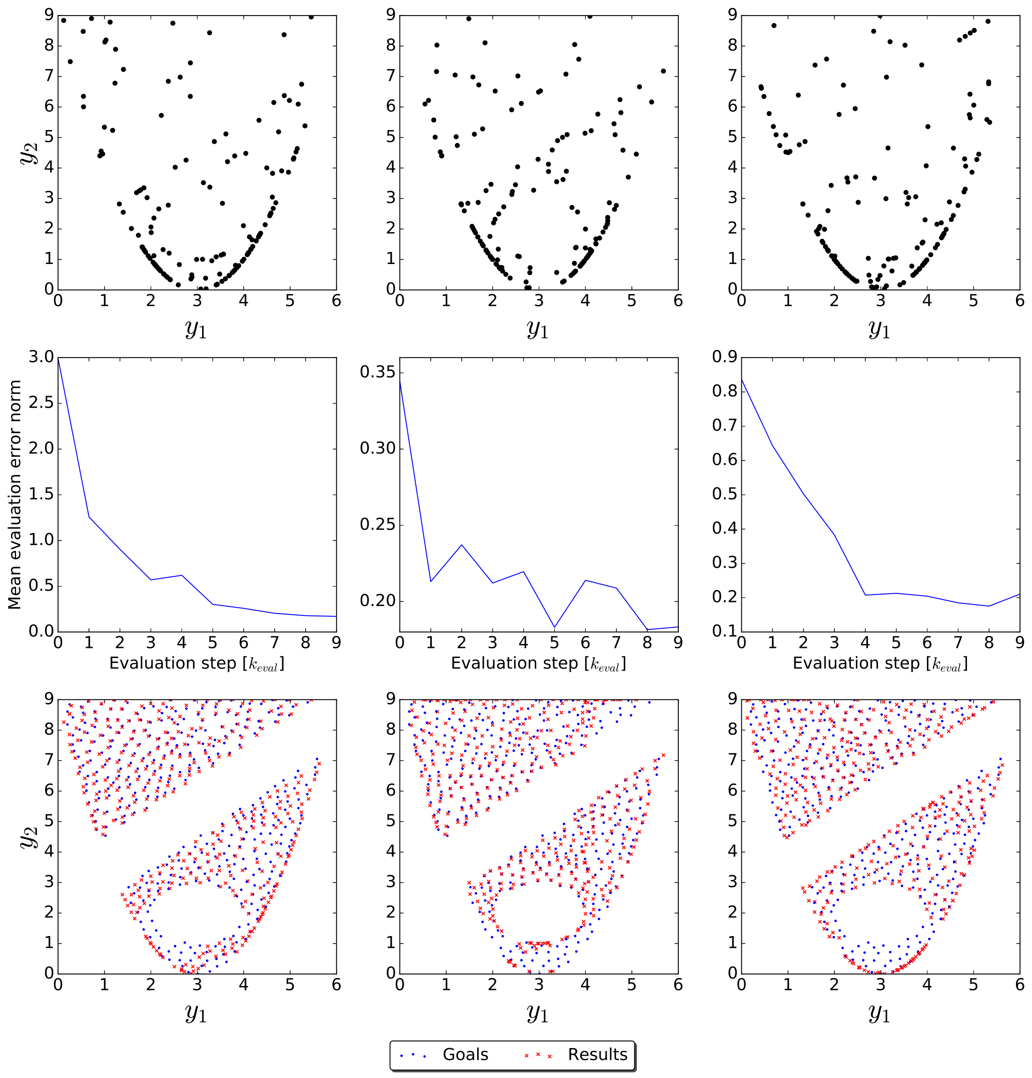

Figure 4. Using our incremental learning approach to solve the inference problem of input commands from output goals. The columns correspond to experiments using different random seeds. The first row are the training samples used along all the training steps. The second row corresponds to the evolution of the average evaluation error norm at each training step. The third row corresponds to the evaluation results after the last training step.

generated obtaining 15 random inputs x from the allowed input space and obtaining its corresponding output y. Each model is trained with 10 data batches and after each training step models are evaluated. The evaluation data set consists of 177 samples uniformly distributed over the whole sensor space.

In Figure 4, the results for the simulations to learn the parabolic shaped system are shown. Each column correspond to a simulation with each of the three different random seeds, whereas the first row correspond to the accumulated random data points used to train the models. The second row corresponds to the average norm error during the evaluation after each training step. Finally, the third row corresponds to the output projection of the system after a final evaluation. The blue points represent each of the output goals in the evaluation data set, and the small red crosses are the actual reached output configurations.

In the third row of Figure 4, we observe a good performance of the model when solving even though few samples are considered for training (150 samples divided in 10 batches). Moreover, the training samples are not uniformly distributed along the output space due to the their random source. From the last row, it is also obvious that the learning system struggles fitting the model to the system around the minimum of the parabola. We argue that it is due to the presence of many constraints in the neighborhood and the non-linear nature of the system.

5. Conclusion

We have presented a novel approach to solve the regression problem for multivariate input-output systems. The illustrative examples show the suitability of the approach to efficiently learn a probabilistic model of the system and update the model with new incoming data batches without the need of keeping in memory the already learned data. On the other hand, we introduced a simple example to demonstrate with good results the applicability of the approach to solve the inference of input commands from desired output goals. The proposed approach is also suitable to solve the prediction problem, in other words it can be used to predict output results from input commands.

We leave for future work three main topics. First, a comparison of our approach with other incremental learning approaches for regression purposes. Second, a deeper study on the parametrization of the model and the study of new divergence measures with more tools to improve the incremental learning mechanism. And third, to apply the approach in more real problems that allows an easy visualization of its performance.

Acknowledgements

J.M. Acevedo-Valle acknowledges the financial support provided by CONACYT grant 216872 and “La Caixa” Foundation International Mobility grant. Karla Trejo acknowledges the financial support provided by CONACYT.

References

[1] Abdelhamid Bouchachia and Charlie Vanaret. Incremental learning based on growing gaussian mixture models. In *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, volume 2, pages 47–52. IEEE, 2011.

[2] Clément Moulin-Frier, Sao M Nguyen, and Pierre-Yves Oudeyer. Self-organization of early vocal development in infants and machines: the role of intrinsic motivation. *Frontiers in psychology*, 4, 2013.

[3] Juan M. Acevedo-Valle, Cecilio Angulo, Nuria Agell, and Clement Moulin-Frier. Proprioceptive feedback and intrinsicmotivations in early-vocal development. In *18th International Conference of the Catalan Association for Artificial Intelligence*. IOS Press, 2015.

[4] Juan M. Acevedo-Valle, Cecilio Angulo, and Clement Moulin-Frier. Autonomous discovery of motor constraints in an intrinsically-motivated vocal learner. *IEEE Transactions on Cognitive and Developmental Systems*, 2017, [In press].

[5] C. Chen, N. Zhang, S. Shi, and D. Mu. An efficient method for incremental learning of gmm using cuda. In *2012 International Conference on Computer Science and Service System*, pages 2141–2144, Aug 2012.

[6] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[7] Paulo Martins Engel and Milton Roberto Heinen. Incremental learning of multivariate gaussian mixture models. In *Brazilian Symposium on Artificial Intelligence*, pages 82–91. Springer, 2010.

[8] Sylvain Calinon. *Robot programming by demonstration*. EPFL Press, 2009.

[9] Jacqueline Gottlieb, Pierre-Yves Oudeyer, Manuel Lopes, and Adrien Baranes. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*, 17(11):585–593, 2013.