Naturally Constrained Online Expectation Maximization

Daniela Pamplona, Antoine Manzanera

Naturally Constrained Online Expectation Maximization

Daniela Pamplona
daniela.pamplona@ensta-paris.fr
U2IS, ENSTA Paris,
Institut Polytechnique de Paris,
828 Boulevard des Maréchaux,
91120 Palaiseau, France

Antoine Manzanera
antoine.manzanera@ensta-paris.fr
U2IS, ENSTA Paris,
Institut Polytechnique de Paris,
828 Boulevard des Maréchaux,
91120 Palaiseau, France

***Abstract*—With the rise of big data sets, learning algorithms must be adapted to piece-wise mechanisms to tackle large-scale calculations' time and memory costs. Furthermore, for most learning embedded systems, the input data are fed sequentially and contingently: one by one, and possibly class by class. Thus, learning algorithms should not only run online but cope with time-varying, non-independent, and non-balanced training data for the system's entire life. Online Expectation-Maximization is a well-known algorithm for learning probabilistic models in real-time, due to its simplicity and convergence properties. However, these properties are only valid in the case of large, independent and identically distributed samples. In this paper, we propose to constrain the online Expectation-Maximization on the Fisher distance between the parameters. After presenting the algorithm, we make a thorough study of its use in Probabilistic Principal Components Analysis. First, we derive the update rules, and then we analyze the effect of the constraint on major problems of online and sequential learning: convergence, forgetting and interference. Furthermore, we use several algorithmic protocols: iid *vs* sequential data, and constraint parameters updated step-wise *vs* class-wise.**

Our results show that this constraint increases the convergence rate of online Expectation-Maximization, decreases forgetting and slightly introduces positive transfer learning.

I. Introduction

As the size of data sets grows, learning paradigms have to process them in a piece-wise manner, in order to decrease the time and memory of large scale calculations. Furthermore, many autonomous systems (e.g. self-driving vehicles, humanoid robots and drones) must adapt to their environment continuously and online. In summary, online incremental learning is becoming a frequent requirement when designing intelligent systems; consequently, new algorithm development questions are being raised. Moreover, in many situations, the input data are presented in a sequential way, where consecutive samples are correlated, thus in this case learning should not only be online but also incremental [1].

In online learning, a well-defined consolidation paradigm increases the learning velocity or reduces the number of input samples needed for a good performance. In a sequential learning framework, consolidation might increase learning efficiency and mitigate catastrophic forgetting, hence avoiding re-learning of classes or a dramatic drop in performance [2]. Another aspect to consider in incremental learning is the interference of passed learned classes with future ones' learning process. First, the systems should be able to learn new classes throughout their life, thus the consolidation process should not impede progress and improvement over time. Second, ideally, systems should use previously learned skills to boost the learning of new ones, thus past knowledge should transfer to future one [3].

Although the memory consolidation problem, particularly the question of catastrophic forgetting, has been posed for more than 30 years (see, for instance, [4], [5]) and thoughtfully studied in the field of neuroscience [6], it is only recently - due to technological advances - that it has gained attention from the machine learning community. There are mainly three approaches to model incremental learning. i) Architecture based, where the system is divided into reusable parts that are less prone to changes and other parts devoted to individual tasks. They are typically *ad hoc* designed and suffer of the scalability problem (see, for instance, [7], [8]). ii) Memory based, where the system relies on episodic memory to store or generate data from previous tasks, which might suffer again of the scalability problem or might not be possible for many autonomous systems (see, for instance, [9]). iii) Regularization based, where the system protects parameters against radical changes when a new class arrives, which typically introduce more parameters and require a high level of hand tuning (see, for instance, [10], [11]). There are many other interesting works on regularization methods in the context of neural networks; however, much less attention has been paid to the context of probabilistic latent variables. Under Bayesian approaches, [12] introduced the variational continual learning, which was later combined with an automated the architecture building process in [13] and [14] incrementally matches the moments of the posterior of a Bayesian neural network. For an in-depth review of the state of the art, see [13].

Probabilistic latent variables models are commonly used in machine learning and applications. They provide a way to model the data taking into account unobserved variability and measurements' noise. In the case of online latent models, a stochastic approximation procedure based on the Fisher metrics was proposed by [15]. Despite its guarantees of con-

vergence in probability under the usual regularity conditions, this method is inefficient since at each time step it is necessary to update and invert a matrix (the Fisher information matrix), whose size quadratically depends on the number of parameters. In [16] was proposed an Online Expectation-Maximization (online EM) method that is guaranteed to converge in probability and - in most usual cases - able to run in real-time. However, this method assumes independence between samples and only converges after many samples; thus, it is not suitable for an embedded incremental learning set-up.

In this paper, we propose a method of constraining the update of online learned parameters using Expectation-Maximization (EM) called Naturally Constrained Online Expectation-Maximization (NAT-oEM). Based on [9], we propose that, at each time step, model parameters should not change radically in a statistical sense, either with respect to consecutive parameters or with respect to last class optimal parameters. We analyze this constraint on Probabilistic Principal Components Analysis under several protocols: (i) the (in)dependence of the input data and (ii) the updating policy of the constraint: class-wise *vs* step-wise, which corresponds to whether the system knows (or not) when the learning class changes.

II. Background

A. The Natural Constraint

In the case of neural networks, the most used method for online learning is the Stochastic Gradient Descent (SGD) [17], which is very simple and robust under Independent and Identically Distributed (iid) samples, but not in the case of sequential learning [18]. In the seminal work of [18], an algorithm has been proposed to reduce the catastrophic forgetting in incremental learning. This method, called Elastic Weights Consolidation (EWC), is based on the idea of constraining the difference between parameters with respect to their importance for the model. Built on the idea of natural learning of [19], they proposed to use the diagonal of the Fisher information matrix to prevent the parameters from changing too strongly (in the Riemann space) from the optimal parameters to the previous tasks, thus avoiding catastrophic forgetting.

One problem of EWC is the lack of scalability: it is necessary to save all the optimal parameters for all the tasks; therefore, the system may run out of memory in an incremental learning set-up. To respond to this problem, online EWC was proposed by [9]. This method uses a Laplacian approximation of the Fisher Information matrix so that the number of parameters for consolidation is constant during learning. Notably, this method solves the scaling problem of EWC without significantly reducing its performance [20].

B. Online Expectation-Maximization

Expectation-Maximization is a standard iterative algorithm introduced by [21], to estimate the parameters θ that maximize the observable data log likelihood $\mathcal{L}(X;\theta)$. It is used, for instance, when the maximum likelihood estimation is intractable or in the presence of latent variables. It assumes the existence of latent variables Z distributed according to $f(Z,\theta)$, a parametric family of probability density functions indexed by a parameter θ. The observation X is then viewed as a function of Z. Since usually it is not given the complete data set $\{X,Z\}$, but only the observable data X, in EM, it is considered instead the expectation of the log likelihood of the complete data set, with respect to the conditional distribution of Z given X.

Each EM iteration is decomposed in two steps: the E-step computes the expectation Q of the data log likelihood (equation 1) and the M-step maximizes Q (equation 2):

$$Q(\theta,\theta_k) = \mathbb{E}_{Z|X,\theta_k}\left[\mathcal{L}(X,Z;\theta_k)\right] \tag{1}$$

$$\theta_{k+1} = \arg\max_{\theta} Q(\theta,\theta_k) \tag{2}$$

In batch mode, as defined in equations 1 and 2, EM is guaranteed to maximize the likelihood function and converges after few iterations, however each iteration is computationally costly [22]. Additionally, in practice the E-step is actually never calculated as defined theoretically, but only the data statistics, here denoted $S_i(X,\theta)$, that are sufficient to calculate the M-step.

More recently, with the development of new big data bases and the gained autonomy of intelligent systems, online algorithms re-gained interest from the machine learning community. However, to our best knowledge, the first online parameter estimation procedure for latent data models was proposed in [15]. In this work, it was proposed to use a stochastic approximation method based on Fisher information matrix $F(\theta)$ to update the parameters at each time step, data point by data point, considering a learning rate γ_k changing over time: [1]

$$\theta_{k+1} = \theta_k + \gamma F_{\theta_k}^{-1} \nabla_\theta \mathcal{L}(x_{k+1};\theta_k) \tag{3}$$

The Fisher information of a parameter θ quantifies the amount of information that an observation x carries about θ. Formally, it is defined as the variance of the score of the observable likelihood function. In case the likelihood is twice differentiable, the Fisher information is equal to the second derivative of the likelihood function. In other words, the Fisher information can be seen as the curvature of the likelihood function (these equalities are summarized in equation 4, see [23] for more details).

$$\mathcal{I}(\theta) = \mathbb{E}_X\left[\left(\frac{\mathrm{d}\mathcal{L}(x;\theta)}{\mathrm{d}\theta}\right)^2\right] = -\mathbb{E}_X\left[\frac{\mathrm{d}^2\mathcal{L}(x;\theta)}{\mathrm{d}^2\theta}\right] \tag{4}$$

Despite its beauty and formal simplicity, in practice this algorithm is computationally intractable when the number of parameters is large. To avoid this problem, [16] proposed an online EM algorithm where the expectation step is replaced by a stochastic approximation step while keeping the maximization step unchanged. Thus, in the online version, the sufficient statistics S^i of the E-step are updated stochastically using a

[1]For the sake of simplicity, in the remaining of the text we will simply refer to $S_i(X,\theta)$ as S_i, $F(\theta)$ as F, and γ_k as γ.

function s^i depending on only one data point x and θ. This means that, in the online version, the equation 5 is used to estimate the E-step statistics, while the equation 2 remains unchanged.

$$S^i_{k+1} = S^i_k + \gamma \left(s^i - S^i_k\right) \tag{5}$$

The online EM algorithm proposed by [16] does, in fact, reduce the computational cost of the stochastic algorithm proposed by [15] and it has similar properties of convergence. However, this convergence is usually slow and it is not valid for the case of sequential data, where the subsequent observations are not independent.

III. Naturally Constrained Online Expectation-Maximization

Based on the Online Elastic Weights Consolidation (online EWC) method, we propose that the update of the model parameters in the online expectation-maximization method proposed by [16] is constrained by their uncertainty. In other words, to consolidate the knowledge previously learned, at each M-step, the new parameters should be constrained to avoid large deviations, in a statistical sense, from the previously learned ones. As proposed in [18], we approximate the Fisher information matrix F by its diagonal matrix, reducing the quadratic dependency to a linear one.

A. Overview

Formally, we propose that the M-step defined in equation 2 is now constrained by the difference between the current new parameter and a reference one, θ^*, on the Riemann space generated by the reference:

$$\theta_{k+1} = \arg\max_{\theta} \left(Q(\theta_k, \theta) - \beta||\theta - \theta^*||_{F^*}\right) \tag{6}$$

where β is a parameter that controls the weight of the constraint. The second term can be maximized using gradient ascent in the Riemann space induced by F.

Then finally, the NAT-oEM is defined in three steps: E-step (as defined by [16] in equation 5), M-step (as defined in the usual EM in equation 2 and R-step as defined below (equation 7)

$$\theta_{k+1\_reg} = \theta_{k+1} - \beta F^{-1} \left(\theta_{k+1} - \theta^*\right) \tag{7}$$

We distinguish between two experimental set-ups: a) the system *does not know* when the learning class is changing. In this case, the reference parameters are equal to the previous iteration parameters, and the constraint is defined on the Fisher information of these parameters. Specifically, this means that for each step k, θ^* is set to θ_{k-1} and F^* is set to $F_{\theta_{k-1}}$. b) the system *knows* when the learning class is changing. In this case, the reference parameters and the Fisher matrix are updated only when the learning class changes (i.e. θ^* is set to θ_{k-1} and F^* is set to $F_{\theta_{k-1}}$ if and only if x_{k-1} was the last data point of a class).

B. NAT-oEM for PPCA

Although proposed a long time ago, Principal Component Analysis (PCA) is still one of the most used algorithms in machine learning due to its simplicity and efficiency [24]. Furthermore, its probabilistic version (Probabilistic Principal Components Analysis (PPCA)) is well known to be very robust in the case of missing data [25].

We denote Z the set of latent variables, which are independent samples of a standard Gaussian distribution. These variables are transformed linearly by the matrix W and an additive component μ, both unknown. Furthermore, this transformation is corrupted by an additive Gaussian noise ε, with zero mean and covariance matrix proportional to the identity matrix. X, the set of the observations, is then formally given by:

$$\begin{aligned} X &= WZ + \mu + \varepsilon \\ Z &\sim \mathcal{N}(0, \mathbb{I}) \\ \varepsilon &\sim \mathcal{N}(0, \sigma^2\mathbb{I}) \end{aligned} \tag{8}$$

Under this model, X is also Gaussian distributed, with mean μ and covariance $C = WW^T + \sigma^2\mathbb{I}$. The final goal is to estimate the parameters $\theta = W, \mu, \sigma^2$.

If we denote $M = W^TW + \sigma^2\mathbb{I}$, for an observable x, the expected latent variable z is given by $z = M^{-1}W(x - \mu)$. In PPCA there are 4 sufficient statistics [26], which are evaluated on a single point x and the constrained parameters of the previous iteration μ, M, W:

$$s^0(x, \mu) = (x - \mu)^T(x - \mu) \tag{9}$$

$$s^1(x, z) = (x - \mu)z^T \tag{10}$$

$$s^2(x, \sigma, M) = \sigma^2 M^{-1} + zz^T \tag{11}$$

$$s^3(x) = x \tag{12}$$

In the E-step these statistics are stochastically integrated as in equation 5. The M-step is then defined with respect to the sufficient statistics S^i:

$$\mu_{k+1} = S^3 \tag{13}$$

$$W_{k+1} = S^1 S^{2^{-1}} \tag{14}$$

$$\begin{aligned} \sigma^2_{k+1} = \frac{1}{d}(&S^0 - 2\,\mathrm{Tr}(S^1W^T) \\ &+ \mathrm{Tr}(S^2W^TW)) \end{aligned} \tag{15}$$

And the R-step is then defined with respect to the updated parameters, here marked with *:

$$\mu_{k+1\_reg} = \mu_{k+1} - \beta F^{*^{-1}}_{\mu} \left(\mu_{k+1} - \mu^*\right) \tag{16}$$

$$W_{k+1\_reg} = W_{k+1} - \beta F^{*^{-1}}_{W} \left(W_{k+1} - W^*\right) \tag{17}$$

$$\sigma^2_{k+1\_reg} = \sigma^2_{k+1} - \beta F^{*^{-1}}_{\sigma^2} \left(\sigma^2_{k+1} - \sigma^{2^*}\right) \tag{18}$$

If we denote $[i, j]$ the entry cell (i, j) of matrices, additionally $\mathbb{J}_{i,j}$ is the single entry matrix, a matrix of zeros except for $\mathbb{J}_{i,j}[i, j] = 1$ (see [27] for more details), Tr and Diag

are the trace and diagonal matrix operators, then the Fisher information matrices are defined as follows [2]:

$$F_\mu = \mathrm{Diag}(C^{-2}) \tag{19}$$

$$F_{W_{[i,j]}} = \mathrm{Tr}((C_k^{-1}(W\mathbb{J}_{i,j}^T + \mathbb{J}_{i,j}W^T))^2) \tag{20}$$

$$F_{\sigma^2} = \mathrm{Tr}(C^{-1}) \tag{21}$$

IV. Experimental Details

A. Input Data

For the PPCA experiment we used synthetic data, generated directly by the model as described below.

Dataset generation: We set $d = 16$ and $q = 3$, and generated a matrix R of size $d \times q$ with random values uniformly distributed between -0.5 and 0.5. To ensure that the transformation matrix columns were independent, we set W equal to the left matrix of the singular value decomposition of the matrix R. In this case, we set $\mu = 0$, and σ to a random value between 0 and 1. Thereafter, we generated the latent variables Z using the pseudo-random numbers generator of *NumPy*. And finally, observations were generated according to the model described in equation 8. Following this process we generated 10^6 training data points and 10^6 testing data points.

Class assignment: The definition of the concept "class" under the PPCA generative model is not trivial since there are no classes explicitly defined. Therefore we defined a criterion to assign data points to classes that is not trivial and does not depend directly on the model parameters. We started by setting the number of classes to 4, class A, B, C and D. For each class, a centroid was randomly selected from the training data and added some noise. Finally, each of the data points (training and testing) was assigned to the class whose centroid is closest, in a Euclidean sense.

B. Training and Testing Data Sets and Meta-parameters

Each condition was run five times with different initializations. All the figures refer to the average log-likelihood over the five runs. The number of data points for training and testing was the same for all three experimental set-ups. In the online learning the points were fed to the algorithm independently of their class. In the sequential learning, the algorithm was fed class by class. Tab. I shows the exact number of data points for training and testing. Both meta-parameters γ and β depended on the iteration number k and, after running the algorithm one time, the value that raised higher log-likelihood was selected. As shown in [26], the exponent of γ must lie within $]0.5, 1[$, we tested for 0.6 and 0.9 in the online unconstrained case. Also in [26] is shown that the multiplicative scalar of this parameter must be within $]0.5, 1[$; we selected the higher decimal value that allowed the algorithm to converge. The exponent of β was defined to agree with the exponent of γ and the multiplicative scalar was tested for 10^i with $i = [\![-2, 2]\!]$ for each condition separately.

[2]The detailed derivation of the Fisher Information matrix for the PPCA is available in the appendix.

TABLE I
Training and testing data sets sizes and meta-parameters.

N Training	3200
N Testing	800
γ online EM	$0.9k^{-0.9}$
γ NAT-oEM-step	$0.9k^{-0.9}$
γ NAT-oEM-class	$0.5k^{-0.9}$
β	$k^{-0.9}$

V. Results

A. NAT-oEM with iid Samples

We start this section with the case where input data are fed to the system independently of the class. This corresponds to the conditions where online EM was designed: iid samples. Evidently, in this case, there is no catastrophic forgetting; this case's interest is to measure the convergence of online EM with the natural constraint. Furthermore, since we are not in the case of sequential learning, the Fisher information matrix is updated at each time step.

In Fig. 1 which shows the evolution of the testing data average log-likelihood over time, it is clear that for the same number of iterations, the average log-likelihood of NAT-oEM is higher than the online EM. After 500 iterations, the likelihood lines become parallel, which means that the log-likelihood difference between the constrained and unconstrained case is constant. Thus, the natural constraint is a more efficient solution than usual online EM.

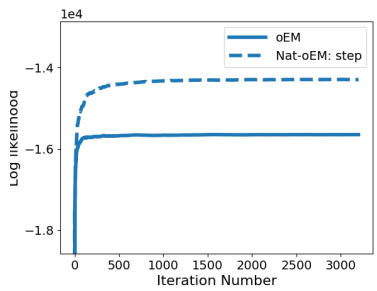

Fig. 1. Average log-likelihood of the testing dataset in function of the learning iteration in the case of iid samples. From the initial iterations the average log-likelihood is higher in the constrained case and the difference between the two log-likelihoods is constant.

B. NAT-oEM with Sequential Samples

In the most real world learning systems, the input data are generally presented class by class, so consecutive training samples are not independent. Under this scenario, we consider two updating policies of the reference Fisher information matrix and parameters: i) NAT-oEM-step: they are updated at each time step and ii) NAT-oEM-class: they are only updated at the end of each class.

On the one hand, NAT-oEM-step requires many computations, so it is fairly slow. On the other hand, NAT-oEM-class requires the system to be notified of each learning class's end. This information might not be available in all the systems,

although it might sometimes be possible to detect it from the data's basic statistics. Besides, in the case of embedded systems like robots, autonomous vehicles, or drones, the transition between classes is not clearly defined. For instance, for visual data, a bathroom is clearly from a different class of a kitchen, but since a domestic robot does not jump between rooms, the transitions are smooth and hard to define.

In the case of sequential learning, there are three major problems to be addressed: i) the possibility of the system getting stuck in local maximum, ii) the possibility of the system forgetting what has been learned, iii) the possibility that learned classes interfere negatively with the ability of the system learning new tasks.

To investigate problem i) we measure the average log-likelihood over the entire test data set, even for the classes that were not learned. If the algorithm is stuck in a local maximum, then the average log-likelihood does not increase over time. Fig. 2 shows the average log-likelihood over time for the three cases: online EM, NAT-oEM-step and NAT-oEM-class. In the case of unconstrained learning, the average log-likelihood behaves erratically and can decrease over time (during the learning of class C). This problem is a direct consequence of sequential samples. In the case of non iid samples, the guarantees of convergence of online EM are not any more valid; therefore, this problem may arise frequently. In the case of constrained learning with the policy of updating the Fisher information matrix at each step (NAT-oEM-step), the average log-likelihood increases over time, the learning curve is abrupt at the beginning of a new class furthermore, the difference between the likelihood of this constrained policy and unconstrained increases over time.

Finally, in the case of constrained learning with the policy of updating the Fisher information matrix only by the end of each class (NAT-oEM-class), the average log-likelihood increases steadily and slowly over time. However, the average log-likelihood by the end of learning is similar to the unconstrained case.

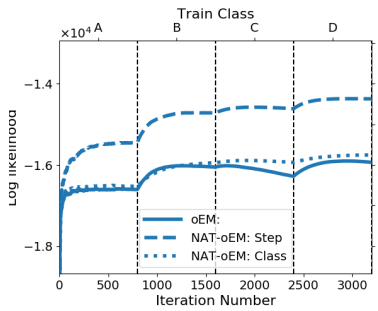

Fig. 2. Average Log-likelihood of the test data set in function of the learning iteration in the case of sequential samples. The beginning of each class is marked with a vertical dashed lines. The average log-likelihood of the online EM is the lowest and it behaves erratically: it decreases when learning the class C. The average log-likelihood of the NAT-oEM-step always increases over time and, after some initial iterations, it is always higher than the online EM. The average log-likelihood of the NAT-oEM-class is always crescent, but in a rather slow way.

To study problem ii) we measure the average log-likelihood of each class independently during the sequential learning process. If the log-likelihood of a class decreases when later classes are learned, then we say that the algorithm has forgotten what it has learned. This is measured in the Fig. 3. The end of each class learning sequence is marked with vertical dashed lines. For each class, the test data average log-likelihood is shown on the columns. The log-likelihood is calculated from the beginning of the learning sequence until the end of the learning process. For the online EM case, there is clear forgetting of some classes, namely, the average log-likelihood of A decreases when B and C are learned, and the average log-likelihood of C decreases when D is learned. For the NAT-oEM-step case, there is no visible forgetting, except for class A. Finally, for the NAT-oEM-class case, there is no visible forgetting, even for class A.

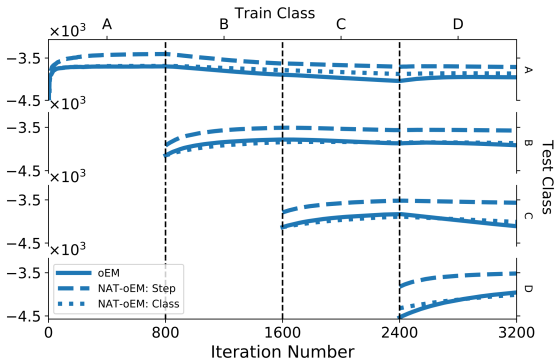

Fig. 3. Class by class average Log-likelihood of the test data set in function of the learning iteration. The beginning of each class is marked with a vertical dashed line. Rows correspond to the average log-likelihood of each class. For the three cases online EM, NAT-oEM-step and NAT-oEM-class, there is forgetting of class A, although stronger in the unconstrained case. In the online EM there is also forgetting of C while learning D. In general the natural constraint reduces forgetting episodes.

In order to analyze the impact of problem iii) we measure, for each possible pair of classes (i, j), the log-likelihood of j starting from the output of learning i, and compare it with the likelihood of learning j from scratch with random initialization. Fig. 4 shows the three "interference matrices", each corresponding to the case of online EM, NAT-oEM-step and NAT-oEM-class. The matrix reads as follows: each row is the class that was initially learned, and each column is the class that was learned next. Each value of the matrix (i, j) is then equal to the difference of the log-likelihood of the second class j after learning i then j, with log-likelihood of the second class j learned from scratch. If this interference matrix's values are negative, then there was negative interference, meaning the fact that the system learned first i reduced its ability to learn j. If the matrix values are zero, then there was no interference, meaning, the fact that the system learned first i does not change its ability to learn j. If the matrix values are positive, then there was positive interference (also called transfer), meaning that the system learned first i increases its ability to learn j. In the case of online EM, there is only non positive interference: most of the interference matrix values

are zero or close to zero. Thus, the interference is negative or non-existent. In the case of NAT-oEM-step there is a light negative interference and a strong positive interference from class D to A, B, C. Thus, there is a substantial transfer from class D to all the others. In the case of NAT-oEM-class there is a fair negative interference between all the classes, except for the transfer between class C and D. In summary, the positive interference was only possible when the algorithm was naturally constrained, but the negative interference was not avoided even in this case.

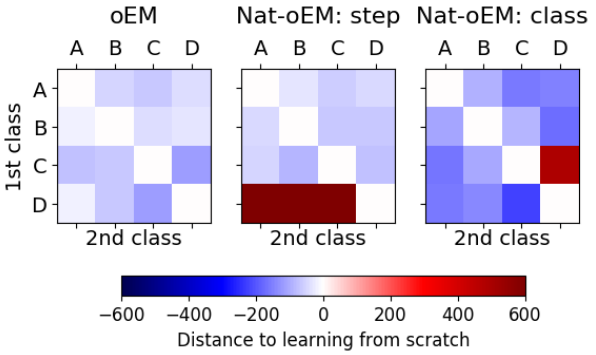

Fig. 4. Interference matrices for the three cases: online EM, NAT-oEM-step and NAT-oEM-class. Each row corresponds to the class initially learned, and each column corresponds to the class learned thereafter. The interference was measured by subtracting the log-likelihood of the second class learned from scratch from the log-likelihood of the second class learned after the first one. In the unconstrained case there is only negative transfer, while some positive transfer was found in the naturally constrained case, both step and class wise.

VI. Conclusions

In this paper, we introduced a constraint for Online Expectation-Maximization based on the Fisher information of the parameters to be estimated, called NAT-oEM. We tested this constraint on PPCA with two scenarios: independent and sequential learning. In the last scenario, we considered two update policies: with and without the information on when each learning class is over. Under these experimental set-ups, NAT-oEM-step was more effective than NAT-oEM-class in terms of convergence and forgetting mitigation, and both more efficient than online EM.

However, in terms of knowledge transfer from previous classes, NAT-oEM was not clearly different from online EM.

In general, it can be seen that the class-wise policy brings benefits with respect to the unconstrained case: it avoids systematically catastrophic forgetting. The step-wise policy is computationally heavier because the parameters are updated every step, but it turns out to be much more effective in terms of online learning, consolidation and transfer. Furthermore, it is also more realistic in terms of natural learning, since the ”learning class signals” may not be available or even not exist at all when the change is too smooth from one class to another.

Due to limited space constraints, it was left out two other problems to address in sequential learning: the number of samples to be used from each class and the class order of learning. Our preliminary results indicate (data not shown) that this study's major properties are kept with other sample sizes. However, as Fig. 4 suggests, changing the class order might positively impact the average log-likelihood.

Possible future work directions might include: to analyze this method with more complex data models, or with more realistic data, and test other policies to update the Fisher information matrix and reference parameters, for instance, using hybrid solutions between the step and class rules like an update every N steps, or exploit (weak) sequence change signals to activate updating.

Acknowledgment

This research has been made under financial support from the French Government Defense Procurement Agency (DGA/AID).

References

[1] A. Gepperth and C. Karaoguz, “A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems,” *Cognitive Computation*, vol. 8, no. 5, pp. 924–934, 2016.

[2] R. M. French, “Catastrophic Forgetting in Connectionist Networks: Causes, Consequences and Solutions,” *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, jan 1999.

[3] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, “Don't forget, there is more than forgetting: new metrics for continual learning,” *arXiv preprint arXiv:1810.13166*, 2018.

[4] R. Ratcliff, “Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions,” *Psychological Review*, vol. 97, no. 2, pp. 285–308, 1990.

[5] M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[6] E. M. Robertson, A. Pascual-Leone, and R. C. Miall, “Current concepts in procedural consolidation,” *Nature Reviews Neuroscience*, vol. 5, no. 7, pp. 576–582, 2004.

[7] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.

[8] O. Ostapenko, M. Puscas, T. Klein, P. Jähnichen, and M. Nabi, “Learning to remember: A synaptic plasticity driven framework for continual learning,” *arXiv preprint:1904.03137*, 2019.

[9] J. Schwarz, J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, “Progress & compress: A scalable framework for continual learning,” *arXiv preprint arXiv:1805.06370*, 2018.

[10] F. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” *Proceedings of machine learning research*, vol. 70, p. 3987, 2017.

[11] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, “Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence,” *arXiv preprint 1801.10112*, no. 1, 2018.

[12] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, “Variational continual learning,” 2018.

[13] T. Adel, H. Zhao, and R. E. Turner, “Continual learning with adaptive weights (claw),” in *International Conference on Learning Representations*, 2020.

[14] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming catastrophic forgetting by incremental moment matching,” in *Advances in neural information processing systems*, 2017, pp. 4652–4662.

[15] D. M. Titterington, “Recursive parameter estimation using incomplete data.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 46, no. 2, pp. 257–267, 1984.

[16] O. Cappé and E. Moulines, “Online EM Algorithm for Latent Data Models,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, pp. 593–613, 2007.

[17] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks," in *International Conference on Learning Representations*, 2014.
[18] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, 2016.
[19] S.-i. Amari, "Natural Gradient Works Efficiently in Learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.
[20] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.
[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.
[22] C. J. Wu, "On the convergence properties of the em algorithm," *The Annals of statistics*, pp. 95–103, 1983.
[23] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.
[24] I. T. Jolliffe, "Principal component analysis and factor analysis," in *Principal component analysis*. Springer, 1986, pp. 115–128.
[25] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2013.
[26] O. Cappé, "Online expectation maximisation," *Mixtures: Estimation and Applications*, pp. 31–53, 2010.
[27] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," *Technical University of Denmark*, nov 2012, version 20121115. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?3274
[28] D. Slepian, "Estimation of signal parameters in the presence of noise," *Transactions of the IRE Professional Group on Information Theory*, vol. 3, no. 3, pp. 68–89, 1954.
[29] W. Bangs, "Array processing with generalized beam-formers," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1972.

VII. Appendix: Fisher Information for PPCA parameters

In this Appendix, we derive the Fisher Information matrix of PPCA used in equations 16 - 21. For the sake of simplicity, we removed all the subscripts k, denoting the parameters/data values at iteration k. Additionally we replaced $F(\theta_k)$ by F.

For multivariate data Gaussian distributed $Y \sim \mathcal{N}(\mu, C)$ depending on some parameters $\theta = [\theta_1, \theta_2, \ldots, \theta_K]$, the Slepian-Bangs formula [28], [29] tells us that the entry m, n of the Fisher information matrix ($\mathcal{I}$) is given by:

$$\mathcal{I}[m,n] = \frac{\mathrm{d}\mu}{\mathrm{d}\theta_m}^T C^{-1} \frac{\mathrm{d}\mu}{\mathrm{d}\theta_n} + \frac{1}{2} Tr(C^{-1} \frac{\mathrm{d}C}{\mathrm{d}\theta_m} C^{-1} \frac{\mathrm{d}C}{\mathrm{d}\theta_n}) \quad (22)$$

With

$$\frac{\mathrm{d}\mu}{\mathrm{d}\theta_m} = \left[\frac{\mathrm{d}\mu_1}{\mathrm{d}\theta_m}, \ldots, \frac{\mathrm{d}\mu_N}{\mathrm{d}\theta_m} \right] \quad (23)$$

$$\frac{\mathrm{d}C}{\mathrm{d}\theta_m} = \begin{bmatrix} \frac{\mathrm{d}C_{1,1}}{\mathrm{d}\theta_m} & \cdots & \frac{\mathrm{d}C_{1,N}}{\mathrm{d}\theta_m} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}C_{N,1}}{\mathrm{d}\theta_m} & \cdots & \frac{\mathrm{d}C_{N,N}}{\mathrm{d}\theta_m} \end{bmatrix} \quad (24)$$

Particularly, in the Probabilistic PCA model defined in 8, Y is Gaussian distributed and the set of parameters $\theta = \left[\mu_1, \ldots, \mu_d, W_{1,1}, \ldots, W_{d,q}, \sigma^2\right]$ has $d + d \times q + 1$ elements. In terms of partial derivatives we have:

$$\frac{\mathrm{d}\mu_j}{\mathrm{d}\mu_i} = \delta_{i,j} \quad (25)$$

$$\frac{\mathrm{d}C}{\mathrm{d}W\,[i,j]} = W \mathbb{J}_{i,j}^T + \mathbb{J}_{i,j} W^T \quad (26)$$

$$\frac{\mathrm{d}C}{\mathrm{d}\sigma^2} = \mathbb{I} \quad (27)$$

Additionally, the expected value of Y only depends on μ and the covariance depends on W and σ. Therefore all other partial derivatives are equal to zero. The non zero values of $\mathcal{I}$ are derived below. For $0 < m, n \leq d$,

$$\mathcal{I}[m,n] = C^{-1}[m,n] \quad (28)$$

For $d < m, n \leq (d \times q)$, gathering equations 22 and 26 we have

$$\mathcal{I}[m,n] = \frac{1}{2} \operatorname{Tr}\left(C^{-1}(W \mathbb{J}_{i,j}^T + \mathbb{J}_{i,j} W^T) C^{-1} (W \mathbb{J}_{k,l}^T + \mathbb{J}_{k,l} W^T) \right) \quad (29)$$

with $m = i + c * j$ and $n = k + c * l$ being the linear indexes of the matrix entry (i, j). For $(d \times q) < m \leq (d \times q + 1)$ and $m' = m - (d \times q)$

$$\mathcal{I}[m, d \times q + 1] = (C^{-2} W)_{m'} \quad (30)$$

For $(d \times q) < n \leq (d \times q + 1)$ and $n' = n - (d \times q)$

$$\mathcal{I}[d \times q + 1, n] = (C^{-2} W)_{n'} \quad (31)$$

And finally,

$$\mathcal{I}[d \times q + 1, d \times q + 1] = \frac{1}{2} \operatorname{Tr}(C^{-2}) \quad (32)$$