Fast Reinforcement Learning with Incremental Gaussian Mixture Models

Rafael Pinto
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Canoas, RS
rafael.pinto@canoas.ifrs.edu.br

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533632

***Abstract*—This work presents a novel algorithm that integrates a data-efficient function approximator with reinforcement learning in continuous state spaces. An online and incremental algorithm capable of learning from a single pass through data, called Incremental Gaussian Mixture Network (IGMN), was employed as a sample-efficient function approximator for the joint state and Q-values space, all in a single model, resulting in a concise and data-efficient algorithm, i.e., a reinforcement learning algorithm that learns from very few interactions with the environment. Results are analyzed to explain the properties of the obtained algorithm, and it is observed that the use of the IGMN function approximator brings some important advantages to reinforcement learning in relation to conventional neural networks trained by gradient descent methods.**

***Index Terms*—Reinforcement Learning, Gaussian Mixture Models, Incremental Learning, Online Learning**

I. Introduction

Reinforcement learning is at the center of many recent accomplishments in artificial intelligence, such as playing Atari games [1] and playing *go* at the grandmaster level [2]. However, most common approaches suffer from low *data efficiency*, which means that a high number of training episodes are necessary for the agent to acquire the desired level of competence on diverse tasks. The Deep Q-Learning Network (DQN) [3] and its variants [4], [5] as well as A3C [6] and other model-free actor-critic or policy gradient methods [7] require millions of agent-environment interactions due to very inefficient learning. This is not acceptable for some classes of tasks, such as robotics, where failure and damage must be minimized. A robot cannot afford to fall from stairs a thousand times before learning to avoid them. More data-efficient algorithms are necessary.

Some model-based solutions to such a problem were proposed, such as [8], [9], but results are still far from ideal, possibly due to its reliance on non-data-efficient function approximators like neural networks trained by gradient descent. While other works focus on the reinforcement learning side itself for data efficiency (by improving on exploration policies [10], for instance), the function approximation side is often neglected.

This research proposes a new solution to this problem, by integrating a sample-efficient function approximator, namely the Incremental Gaussian Mixture Network (IGMN) [11]–[13], with reinforcement learning techniques, reducing the number of required interactions with the real environment. It has been observed before that decoupling representation learning from behavior learning can be beneficial to reinforcement learning with function approximation [14], [15], since representations can be learned much faster than behavior, which relies on sparse reward signal, while some argue that behavior can be learned faster due to slow gradient descent on representation learning [16]. IGMN can learn both representations very fast and make locally linear predictions of Q-values based on those representations, while mostly avoiding catastrophic forgetting in the process.

This work is structured as follows: section II presents related works, while section III presents the latest iteration of the IGMN algorithm. Section IV presents the proposed algorithm by combining IGMN and reinforcement learning. Section V shows experimental results, and section VI finishes this work with discussions and future works.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

II. Related Works

In [17], the authors use online EM to learn a single Gaussian mixture model (GMM) of the joint state-action-Q-value space. The algorithm is incremental, but it starts with some randomly placed Gaussian components instead of an empty model like the IGMN. Its component creation rule is based on an inference error threshold, as well as a Mahalanobis distance criterion. Continuous action selection is done approximately by computing the Q-value of a few random actions and selecting the one with the largest value. Matrix inverses are computed at each step, something that is avoided in the present work by using the fast variant of the IGMN algorithm (FIGMN) [18], [19]. An important contribution from the authors is the use of Q-value estimation variance provided by the GMM to drive exploration, resulting in something reminiscent of Q-value sampling in Bayesian Q-learning [20]. Another important insight is the need to forget old values, as Q-values are non-stationary and learned through bootstrapping, meaning that old values are mostly wrong.

In [21], the IGMN was applied to a traffic simulator with continuous action selection. The algorithm is fed with the current state and the maximum stored Q-value and outputs the greedy action. Moreover, IGMN outputs the action variance too, allowing for exploration by Gaussian noise around the greedy action, with the interesting feature that this variance

is also adaptive, meaning that the exploration rate adapts to each state region. IGMN can linearly generalize inside each component, making the predictions smoother. The present work is targeted at discrete action environments instead of continuous actions, and thus uses a different architecture with one Q-value per action and some additions to the learning algorithm.

Recent works on episodic memory for reinforcement learning, such as Model-Free Episodic Control (MFEC) [16] and Neural Episodic Control (NEC) [22], have introduced a framework for stable, long-term, and immediately accessible memories to be stored and used analogously to Q-tables. Those models allow for generalization through k-Nearest Neighbors (kNN) search or by the use of a Radial Basis Function (RBF) layer and a linear output layer. An issue with both methods is the rapid filling of the memory, which is dealt with by removing the least recently used entries. In [23], authors propose to cluster the least recently used entry with its nearest entry, avoiding the removal of rare but important memories. That shows some resemblance to the inner workings of the IGMN algorithm, as it also stores memories as they arrive, available for immediate contribution to inference. Three main differences are: a) IGMN clusters memories continuously: there is no wait for the buffer to fill up (we observed better models constructed in this way; this feature was also added to MFEC in [24]); b) IGMN memory grows and shrinks as necessary; c) memories are complete Gaussian distributions of the joint input-output space, allowing for feature selection and linear regression inside each memory unit (this avoids the step-wise behavior observed in function approximators like kNN and RBF neural networks and requires fewer memory units). A downside of IGMN is the $\mathrm{O}(N^2)$ time complexity on the number of dimensions.

It is also worth noting that other sample-efficient function approximators do exist, such as tile-coding [25]. However, this approach employs a fixed number of grids with fixed cell sizes covering the entire state-space, while our approach is able to allocate resources to relevant regions. Other online possibilities might include Recursive Least Squares (RLS) [26], but this would restrict the model to linear functions. In fact, an IGMN can be seen as a mixture of locally linear regressors, which gives it the ability to model non-linear data. In that matter, Kernel Recursive Least Squares (KRLS) [27] is a non-linear version of RLS, but it needs to store a large number of example vectors in order to better approximate functions, while performing a single RLS on a projected space spanned by those examples. The IGMN performs multiple local linear regressions, which are then combined into a final result. It also stores compact representations of multiple examples in multivariate Gaussian components, reducing the number of needed components.

III. Incremental Gaussian Mixture Model

In the next subsections we describe the current version of the IGMN algorithm.

A. Learning

The algorithm starts with no components, which are created as necessary (see subsection III-B). Given vector $\mathbf{x}$ (a single instantaneous data point), which includes both input and output concatenated, the IGMN algorithm processing step is as follows. First, the squared Mahalanobis distance $d_M^2(\mathbf{x}, j)$ for each component j is computed:

$$d_M^2(\mathbf{x}, j) = (\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j (\mathbf{x} - \boldsymbol{\mu}_j) \quad \forall j \tag{1}$$

where $\boldsymbol{\mu}_j$ is the j^{th} component mean and $\boldsymbol{\Lambda}_j$ its full precision matrix (the inverse of the covariance matrix) . If any $d_M^2(\mathbf{x}, j)$ is smaller than than $\chi^2_{D,1-\beta}$ (the $1-\beta$ percentile of a chi-squared distribution with D degrees-of-freedom, where D is the input dimensionality and β is a user defined meta-parameter, e.g., 0.1), an update will occur, and posterior probabilities are calculated for each component as follows:

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \exp\left(-\frac{1}{2} d_M^2(\mathbf{x}, j)\right) \quad \forall j \tag{2}$$

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{k=1}^{K} p(\mathbf{x}|k)p(k)} \quad \forall j \tag{3}$$

where $p(j)$ is the component's prior probability (equation 13) and K is the number of components.

After that, according to [18], [19], parameters are updated as follows. Note that rank one updates are applied directly to the precision matrices ($\boldsymbol{\Lambda}$) and matrix inversions are avoided.

$$v_j(t) = v_j(t-1) + 1 \tag{4}$$

$$sp_j(t) = sp_j(t-1) + p(j|\mathbf{x}) \tag{5}$$

$$\mathbf{e}_j = \mathbf{x} - \boldsymbol{\mu}_j(t-1) \tag{6}$$

$$\omega_j = \frac{p(j|\mathbf{x})}{sp_j} \tag{7}$$

$$\Delta\boldsymbol{\mu}_j = \omega_j \mathbf{e}_j \tag{8}$$

$$\boldsymbol{\mu}_j(t) = \boldsymbol{\mu}_j(t-1) + \Delta\boldsymbol{\mu}_j \tag{9}$$

$$\mathbf{e}_j^* = \mathbf{x} - \boldsymbol{\mu}_j(t) \tag{10}$$

$$\boldsymbol{\Lambda}(t) = \frac{\boldsymbol{\Lambda}(t-1)}{1-\omega} + \boldsymbol{\Lambda}(t-1)\mathbf{e}\mathbf{e}^T\boldsymbol{\Lambda}(t-1)\frac{\omega(1-3\omega+\omega^2)}{(\omega-1)^2(\omega^2-2\omega-1)} \tag{11}$$

$$|\boldsymbol{\Sigma}(t)| = (1-\omega)^D |\boldsymbol{\Sigma}(t-1)| \left(1 + \frac{\omega(1+\omega(\omega-3))}{1-\omega}\mathbf{e}^T\boldsymbol{\Lambda}(t-1)\mathbf{e}\right) \tag{12}$$

$$p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q} \tag{13}$$

where sp_j and v_j are the posteriors accumulator (a probabilistic version of a counter) and the age of component j (used for prunning), respectively, and $p(j)$ is its prior probability. Detailed learning steps are shown in algorithms 1 and 2.
where $\boldsymbol{\sigma}_{ini}^2$ represents the initial size of Gaussian components (the variance on each feature). It could be set as a fraction of the variances or sensor ranges for each dimension, if available.

B. Creating New Components

If the update condition in algorithm 1 is not met, then a new component j is created and initialized as shown in algorithm 3.

C. Inference

In IGMN, any element can be predicted by any other element. This is done by reconstructing data from the target elements ($\mathbf{x}_t$, a slice of the entire input vector $\mathbf{x}$) by estimating the posterior probabilities using only the given elements ($\mathbf{x}_i$, also a slice of the entire input vector $\mathbf{x}$), as follows:

$$p(j|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|j)p(j)}{\sum_{q=1}^{M} p(\mathbf{x}_i|q)p(q)} \quad \forall j \tag{14}$$

It is similar to equation 3, except that it uses a modified input vector $\mathbf{x}_i$ with the target elements $\mathbf{x}_t$ removed from calculations. After that, $\mathbf{x}_t$ can be reconstructed using the conditional mean:

$$\hat{\mathbf{x}_t} = \sum_{j=1}^{M} p(j|\mathbf{x}_i)(\boldsymbol{\mu}_{j,t} - \boldsymbol{\Lambda}_{j,it}\boldsymbol{\Lambda}_{j,t}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{j,i}))\,, \tag{15}$$

where $\boldsymbol{\Lambda}_{j,it}$ is the submatrix of the jth component's precision matrix associating the known (i) and unknown (t) parts of the data, $\boldsymbol{\Lambda}_{j,t}$ is the submatrix corresponding to the unknown (t) part only, $\boldsymbol{\mu}_{j,i}$ is the jth's component mean's known part (i) and $\boldsymbol{\mu}_{j,t}$ its unknown part (t). This procedure can be seen in algorithm 4, while the covariance matrix block decomposition is shown in equation 16.

$$\boldsymbol{\Lambda}_j = \begin{bmatrix} \boldsymbol{\Sigma}_{j,i} & \boldsymbol{\Sigma}_{j,it} \\ \boldsymbol{\Sigma}_{j,ti} & \boldsymbol{\Sigma}_{j,t} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{\Lambda}_{j,i} & \boldsymbol{\Lambda}_{j,it} \\ \boldsymbol{\Lambda}_{j,ti} & \boldsymbol{\Lambda}_{j,t} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{\Sigma}_{j,i} - \boldsymbol{\Sigma}_{j,it}\boldsymbol{\Sigma}_{j,t}^{-1}\boldsymbol{\Sigma}_{j,ti})^{-1} & -\boldsymbol{\Sigma}_{j,i}^{-1}\boldsymbol{\Sigma}_{j,it}(\boldsymbol{\Sigma}_{j,t} - \boldsymbol{\Sigma}_{j,ti}\boldsymbol{\Sigma}_{j,i}^{-1}\boldsymbol{\Sigma}_{j,it})^{-1} \\ -\boldsymbol{\Sigma}_{j,t}^{-1}\boldsymbol{\Sigma}_{j,ti}(\boldsymbol{\Sigma}_{j,i} - \boldsymbol{\Sigma}_{j,it}\boldsymbol{\Sigma}_{j,t}^{-1}\boldsymbol{\Sigma}_{j,ti})^{-1} & (\boldsymbol{\Sigma}_{j,t} - \boldsymbol{\Sigma}_{j,ti}\boldsymbol{\Sigma}_{j,i}^{-1}\boldsymbol{\Sigma}_{j,it})^{-1} \end{bmatrix}. \tag{16}$$

Note that it requires a matrix inversion of $\boldsymbol{\Lambda}_{j,t}$, which is used in two different places but can be reused in order to avoid two inversions. Also, since in most applications the number of outputs is much smaller than the number of inputs, this does not impose any huge penalty in complexity. Also note that the original FIGMN paper was missing the first two formulas, but they were derived and presented in [28].

D. Removing Spurious Components

A component j is removed whenever $v_j > v_{min}$ and $sp_j < sp_{min}$, where v_{min} and sp_{min} are manually chosen (e.g., 5.0 and 3.0, respectively). In that case, also, $p(k)$ must be adjusted for all $k \in K$, $k \neq j$, using (13). In other words, each component is given some time v_{min} to show its importance to the model in the form of an accumulation of its posterior probabilities sp_j. We do not use component pruning in this work, since it produced deleterious results when combined with reinforcement learning (rare but still important experiences may be forgotten, such as reaching the goal in the Mountain Car problem). New removal procedures that protect high reward memories must be investigated.

IV. Reinforcement Learning with IGMN

Here we develop a data-efficient (and here we define it empirically) reinforcement learning algorithm for continuous state spaces and discrete action spaces. We argue that other algorithms' inefficiencies come partially from the slow function approximators they use, i.e., neural networks trained by gradient descent. Hence, a data-efficient function approximator should be used instead. The algorithm chosen here is the IGMN, due to its speed and versatility. IGMN can be combined with reinforcement learning in various ways [21], but here we focus on the most successful one, called Unified FIGMN-Q in [29]. For the sake of simplicity, it is called Q-IGMN from now on.

Q-IGMN is achieved by modeling the joint space of states and Q-values (one Q-value for each possible action), as this is IGMN's standard way of doing supervised learning. It allows only discrete actions, but, on the other hand, allows for fast action selection (just input the current state and select the action corresponding to the largest Q-value). Algorithm 5 describes the behavior of this architecture.

V. Experiments and Results

OpenAI Gym [30] has been used as the primary platform for testing the proposed algorithm and comparing it to other popular alternatives. This open-source platform provides ready-to-use reinforcement learning environments and is used to provide a leaderboard-like page[1] comparing the performance of algorithms from different users, including the most common

[1]This feature has since been removed from the platform, but experiment result files are still accessible

Algorithm 1 IGMN Learning

Input: β, $\mathbf{X}$, $\boldsymbol{\sigma}_{ini}^2$ ▷ Creation threshold, dataset and initial variance vector
  $K = 0$, $M = \emptyset$ ▷ Start without any Gaussian component
  $\boldsymbol{\Lambda}_{ini} = (\boldsymbol{\sigma}_{ini}^2 \mathbf{I})^{-1}$, $|\boldsymbol{\Sigma}_{ini}| = |\boldsymbol{\Lambda}_{ini}|^{-1}$ ▷ Diagonal matrix inversion and determinant only
  for all input data vector $\mathbf{x} \in \mathbf{X}$ **do**
    $d_M^2(\mathbf{x}, j) = (\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j (\mathbf{x} - \boldsymbol{\mu}_j), \forall j \in M$ ▷ Squared Mahalanobis distance
    if $\exists j,\ d_M^2(\mathbf{x}, j) < \chi^2_{D,1-\beta}$ **then** ▷ Below $1-\beta$ percentile of a χ^2 distribution
      $update(\mathbf{x})$ ▷ Algorithm 2
    else
      $K \leftarrow K + 1$ ▷ Update number of components
      $M \leftarrow M \cup create(\mathbf{x})$ ▷ Algorithm 3

Algorithm 2 Update Components

Input: $\mathbf{x}$ ▷ Input vector including known and unknown parts
  for all Gaussian component $j \in M$ **do**
    $d_M^2(\mathbf{x}, j) = (\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j (\mathbf{x} - \boldsymbol{\mu}_j)$ ▷ Mahalanobis distance from input $\mathbf{x}$
    $p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\boldsymbol{\Sigma}_j|}} \exp\left(-\frac{1}{2} d_M^2(\mathbf{x}, j)\right)$ ▷ Likelihood
    $p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{k=1}^{K} p(\mathbf{x}|k)p(k)}$ ▷ Posterior
    $v_j(t) = v_j(t-1) + 1$ ▷ Increment age
    $sp_j(t) = sp_j(t-1) + p(j|\mathbf{x})$ ▷ Accumulate posterior
    $\mathbf{e}_j = \mathbf{x} - \boldsymbol{\mu}_j(t-1)$ ▷ Compute error
    $\omega_j = \frac{p(j|\mathbf{x})}{sp_j}$ ▷ Compute learning rate
    $\boldsymbol{\mu}_j(t) = \boldsymbol{\mu}_j(t-1) + \omega_j \mathbf{e}_j$ ▷ Update mean
    $\boldsymbol{\Lambda}(t) = \frac{\boldsymbol{\Lambda}(t-1)}{1-\omega} + \boldsymbol{\Lambda}(t-1)\mathbf{e}\mathbf{e}^T\boldsymbol{\Lambda}(t-1)\frac{\omega(1-3\omega+\omega^2)}{(\omega-1)^2(\omega^2-2\omega-1)}$ ▷ Update precision matrix
    $p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q}$ ▷ Update prior
    $|\boldsymbol{\Sigma}(t)| = (1-\omega)^D |\boldsymbol{\Sigma}(t-1)| \left(1 + \frac{\omega(1+\omega(\omega-3))}{1-\omega} \mathbf{e}^T \boldsymbol{\Lambda}(t-1)\mathbf{e}\right)$ ▷ Update determinant

Algorithm 3 Create Component

Input: $\mathbf{x}$ ▷ Full input vector containing concatenated inputs and outputs
  return new Gaussian component j with $\boldsymbol{\mu}_j = \mathbf{x}$, $\boldsymbol{\Lambda}_j = \boldsymbol{\Lambda}_{ini}$, $|\boldsymbol{\Sigma}_j| = |\boldsymbol{\Sigma}_{ini}|$, $sp_j = 1$, $v_j = 1$, $p(j) = \frac{1}{\sum_{k=1}^{K} sp_k}$

algorithms like Q-learning. The performance is measured by two factors: episodes to solve (data efficiency) and mean reward. Each environment has some goal accumulated reward to reach. More precisely, the agent must obtain an average accumulated reward equal or higher than the goal for 100 consecutive episodes, making it a very robust experiment. Time to solve is defined as the first episode of the successful 100 episode window.

More demanding tasks like the Atari games [31] were avoided due to hardware restrictions, but are planned for future experiments. All experiments were executed on an Intel i7 laptop without access to GPU.

Since the OpenAI Gym platform currently only supports the Python language, this was the language of choice for implementing the final experiments. An existing open-source implementation [2] of the IGMN algorithm in this language was used as a basis for the more scalable FIGMN algorithm implementation.

The mountain car task consists of controlling an underpowered car to reach the top of a hill. It must go up the opposite slope to gain momentum first. The agent has three actions at its disposal, accelerating it leftwards, rightwards, or no acceleration at all. The agent's state is made up of two features: current position and speed. Only 200 steps are available for the agent to explore during each episode. This task is considered solved after 100 consecutive episodes with an average of 110 steps or less to reach the top of the hill.

[2] https://github.com/renatopp/liac/blob/master/liac/models/igmn.py

Algorithm 4 Inference (recall)

Input: $\mathbf{x}_i$ ▷ Input vector containing only the known elements

$\boldsymbol{\Sigma}_{j,i}^{-1} = \boldsymbol{\Lambda}_{j,i} - \boldsymbol{\Lambda}_{j,it}\boldsymbol{\Lambda}_{j,t}^{-1}\boldsymbol{\Lambda}_{j,ti}, \quad \forall j \in M$ ▷ Inverse of input portion

$|\boldsymbol{\Sigma}_{j,i}| = |\boldsymbol{\Sigma}_j||\boldsymbol{\Lambda}_{j,t}|, \quad \forall j \in M$ ▷ Determinant of input portion

$d_M^2(\mathbf{x}_i, j) = (\mathbf{x}_i - \boldsymbol{\mu}_{j,i})^T \boldsymbol{\Sigma}_{j,i}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{j,i}), \quad \forall j \in M$ ▷ Squared Mahalanobis distance

$p(\mathbf{x}_i|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\boldsymbol{\Sigma}_{j,i}|}} \exp\left(-\frac{1}{2}d_M^2(\mathbf{x}_i, j)\right), \quad \forall j \in M$ ▷ Likelihoods

$p(j|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|j)p(j)}{\sum_{k=1}^{\sum_{k\in M} p(\mathbf{x}_i|k)p(k)}}, \quad \forall j \in M$ ▷ Posteriors

return $\sum_{j=1}^{j\in M} p(j|\mathbf{x}_i)(\boldsymbol{\mu}_{j,t} - \boldsymbol{\Lambda}_{j,it}\boldsymbol{\Lambda}_{j,t}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_{j,i}))$ ▷ Weighted conditional mean

Algorithm 5 Q-IGMN Algorithm

Input: A is a set of actions, D is the number of dimensions in the state space, γ is the discount factor, ϵ is the exploration rate, E_{max} is the maximum size of the experience replay stack

Initialize IGMN

Initialize empty experience replay stack E with arbitrary maximum size E_{max}

Observe current state $\mathbf{s}$ ▷ Initial state

repeat

  if IGMN is empty or $U(0,1) < \epsilon$ **then** ▷ ϵ-greddy policy

    $a \leftarrow$ random action from A ▷ Exploration

  else

    $Q[\mathbf{s}, .] \leftarrow$ IGMN.recall($\mathbf{s}$) ▷ obtain Q_a for each action in A

    $a \leftarrow argmax_{a'} Q[\mathbf{s}, a']$ ▷ Exploitation

  perform action a and observe reward r and state $\mathbf{s}'$

  push $\{\mathbf{s}, a, r, \mathbf{s}'\}$ into E

  if E is full or episode ended **then** ▷ Learning happens here

    while E not empty **do**

      pop $\{\mathbf{s}, a, r, \mathbf{s}'\}$ from E ▷ Extract last experience from stack

      $Q'[\mathbf{s}', .] \leftarrow$ IGMN.recall($\mathbf{s}'$) ▷ Compute Q' for next state

      $a_{max} \leftarrow argmax_{a'} Q'[\mathbf{s}', a']$ ▷ Greedy action for next state

      $Q_a \leftarrow null, \forall a \in A$ ▷ Create vector of $nulls$

      $Q_{a_{max}} \leftarrow r + \gamma Q'[\mathbf{s}', a_{max}]$ ▷ Target computed according to greedy action

      $\mathbf{x} \leftarrow \{s_1, s_2, ..., s_D, Q_1, Q_2, ..., Q_{|A|}\}$ ▷ Concatenate $\mathbf{s}$ and $Q[\mathbf{s}, .]$

      IGMN.update($\mathbf{x}$) ▷ $null$ values do not update IGMN parameters

  $\mathbf{s} \leftarrow \mathbf{s}'$

until termination

The cart-pole task consists of balancing a pole above a small car that can move left or right at each time step. Four variables are available as observations: current position and speed of the cart and current angle and angular velocity of the pole. Version 0 requires the pole to be balanced for 200 steps, while version 1 requires 500 steps. This task is considered solved after 100 consecutive episodes with an average of 195 steps for version 0 and 475 steps for version 1 without dropping the pole.

Finally, the acrobot task requires a 2-joint robot to reach a certain height with the tip of its "arm". Torque in two directions can be exerted on the 2 joints, resulting in 4 possible actions. The current angle and angular velocity of each joint are provided as observations. There are 200 steps per episode available for exploration. This task is considered solved after 100 consecutive episodes with an average of 100 or fewer steps to reach the target height.

The Q-IGMN algorithm was compared to other 3 algorithms with high scores on OpenAI Gym: Sarsa(λ) with Tile Coding, Trust Region Policy Optimization [32] and Dueling Double DQN [4]. These algorithms were chosen according to the OpenAI Gym rank at the time of writing. Table I shows the number of episodes required for each algorithm to reach the required reward threshold for the 3 tasks [3].

Albeit being very difficult to tune, Q-IGMN proved to be the most data-efficient reinforcement learning algorithm in this set of experiments. It was able to solve each of the tasks in a few

[3]The OpenAI Gym platform computes results based only on successful runs

TABLE I: Number of episodes to solve each task.

Environment	Q-IGMN [4]	Sarsa(λ) [5]	TRPO [6]	Duel DDQN [7]
Cart-Pole V0	**0.5** $\pm$ 0.56	557	2103.50 $\pm$ 3542.86	51.00 $\pm$ 7.24
Mountain Car V0	**0.0**	1872.50 $\pm$ 6.04	4064.00 $\pm$ 246.25	-
Acrobot V0*	**0.0**	742	2930.67 $\pm$ 1627.26	31

* This task is not available in the OpenAI Gym server anymore, so the result can be verified only locally.

episodes. Another interesting result is that Q-IGMN solved most of the tasks with a single Gaussian component, implying that their Q-value function is (at least approximately) linear. The mountain car task, on the other hand, required 8 Gaussian components (its Q-value function has a spiral surface).

However, a single trick was essential to guarantee this data-efficiency: we employed a kind of experience replay buffer. But instead of sampling it randomly and repeatedly at each time step (as commonly done in most works with neural networks), it was sampled from the most recent observation to the oldest one (randomly sampling the experience replay buffer also works here, but performance degrades), *in a single pass* (which means that we still perform only one update per step, they are just shifted and accumulated), and learning only happens when the buffer is full; after that, it is emptied again. Interestingly, time-correlated data does not impair IGMN's performance as it does with neural networks. It is shown in the original Incremental Gaussian Mixture Model paper [33] that data should vary slowly (i.e., it should not be independent and identically distributed (i.i.d.), exactly the opposite condition for neural networks). From another point-of-view, this could be seen as mini-batch learning. This technique drastically improved the algorithm, and there seem to be 2 effects taking place to explain this:

- First, it is common knowledge that conventional Q-learning with function approximation diverges due to the non-stationary nature of the Q-values [34]. The Q-learning update rule uses the function approximator itself to provide a target, which changes immediately, while action selection is also done using the same ever-changing approximator. By doing updates in mini-batches, this issue is minimized, as action selection is performed over a stable function approximator. Then, it is updated all at once, and after that, actions can be selected from a stable approximator again. This bears resemblance to Double Q-Learning, where 2 estimators are used to select actions from a stable approximator which is updated once in a while from the second estimator (which updates constantly), except we use a single estimator with sporadic updates instead;
- The second effect produced by this mini-batch approach is similar to trace decays: while conventional Q-learning updates a single state-action per step, trace decays allow us to update a large portion of the state-action history at once. So, when a goal is reached, its value is rapidly propagated backward. The mini-batch approach results in something very much like it: since the most recently visited states are updated first, older states will receive updated values immediately, eliminating the need to visit those states again to "see" the new values. A key difference is that trace decays perform all these updates at *every* time step, resulting in high computational demands.

In general, larger buffer sizes improved the results. For small episodic tasks like the ones presented here, it is enough to set the maximum buffer size as the maximum number of steps per episode for each task, resulting in episodic batch updates (note, however, that IGMN updates are always incremental, i.e., each experience is presented to the algorithm and then discarded). The resulting buffer sizes are vastly smaller than conventional experience replay buffers for deep learning.

Additionally, in the mountain car task, it was necessary to apply other techniques to ensure stability and convergence:

- The first one was to set an independent learning rate α (with its own annealing schedule) for the Q-value variables in the Q-IGMN. It means that equation 7 (reproduced below) should only apply to the state variables when updating the mean and also the precision matrix.

$$\omega_j = \frac{p(j|\mathbf{x})}{sp_j}\,. \tag{17}$$

  When updating the means and precision matrices for variables corresponding to the Q-values, the following equation should be used instead:

$$\omega_j = p(j|\mathbf{x})\alpha\,. \tag{18}$$

  This effectively decouples representation learning speed from behavior learning speed, and is necessary in some cases, since the default IGMN learning rate ω given by the original equation decreases very fast, which may not be appropriate for a given task. Also, the original equation results in the component mean converging to the true mean of all input data, which is expected when dealing with stationary data, but not with non-stationary data as is the case of the Q-values. The same problem was found by [17] and solved in a way that was appropriate for the used algorithm, but it would be analogous to applying a decay rate to sp_j in Q-IGMN, thus making ω decrease slower. It is not the same as the independent learning rate, as it affects all variables and not only the Q-values. In our experiments, the independent learning rate for Q-values was able to find a solution, while sp decay was not.
- The second one was a technique akin to early stopping, which is used in neural networks to avoid overfitting. Here, a reward threshold (in this case -122) was set in order to stop learning. When this threshold is reached, the ϵ exploration parameter, the α learning rate, and the τ

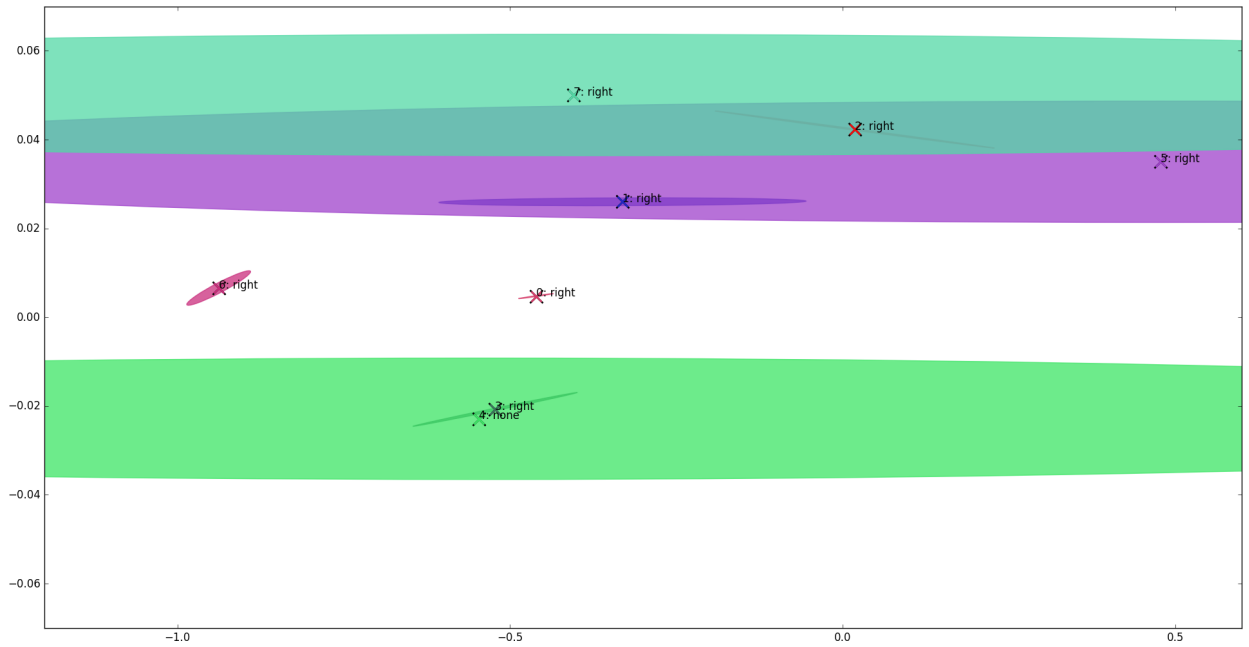

Fig. 1: Policy learned by the Q-IGMN algorithm in the mountain car task. The coloured bands are just very elongated ellipses.

component creation threshold are all set to 0, effectively stopping any learning. This is necessary to avoid new components being created in overlapping positions with old ones, producing catastrophic forgetting. An alternative, which is explored in [28], is to improve the stability of IGMN itself by avoiding overlaps automatically.

The learned policy is shown in figure 1. Actions are shown according to the largest Q-value in the Gaussian means. It is possible to verify that most components suggest the ”right” action, having a single ”none” action on the lower half of the graph. It is expected, as the optimal policy involves applying torque in the same direction as the car's current speed. Upon analyzing the covariance matrices of components #3 and #6, negative covariances were found between the state variables and the ”left” action, meaning that when the car is moving left (negative speed), the Q-value for the ”left” action increases, which matches the expected policy. This is something impossible to achieve with local feature representations like RBF or tile coding, since components only generalize the state space and Q-values do not vary inside a single unit. This explains why Q-IGMN can solve problems with very few Gaussian components (often 1), which also, in turn, makes it very fast.

VI. Conclusion

This work presented the combination of incremental Gaussian mixture models with reinforcement learning. IGMN was employed as a function approximator for Q-values of three classic continuous reinforcement learning tasks. Results show that it presents astounding data-efficiency, learning the three tasks within a few episodes. This can be attributed to the algorithm's similarity to double Q-Learning, delayed Q-Learning, and trace decays, all of which are known to improve data-efficiency, combined with a data-efficient function approximator itself. However, due to time and hardware constraints, the algorithm could only be tested on relatively simple environments, where this performance is not that impressive. New experiments in more complex environments (e.g., Atari) should be conducted in future works to assess the generality and scalability of this solution.

An interesting discovery found while using experience replay is that some drawbacks of conventional neural networks that require some workarounds are not present in IGMN, allowing it to take full advantage of the procedure. For instance, the IGMN does not require i.i.d. data, so experience replay does not need to be sampled randomly. Also, due to its high data-efficiency, only a single scan through the experience replay buffer is necessary. By only updating the model sporadically, simultaneous use of the Q-function estimate for action selection and updating is avoided, improving convergence, which draws parallels with double Q-learning. Thus, one of the main contributions of this research is to show how non-mainstream algorithms can be successfully combined with reinforcement learning, suggesting that neural networks (in their current forms) trained by gradient descent are not the only possibility and possibly not the best.

The main obstacles found reside in the non-stationary nature of the Q-value function, which is not appropriate for the learning rate annealing schedule of IGMN. IGMN finds means of input data, while the Q-value updates require an emphasis on more recent data, in some cases even totally overwriting previous inaccurate values (due to bootstrapping). We fixed this issue by providing a separate learning rate for the Q-values, as well as implementing a tighter control over its variance to avoid singularities.

Another obstacle found during the experiments was IGMN's

high sensitivity to its hyper-parameters (τ and $\boldsymbol{\sigma}_{ini}^2$). A very small change in these parameters is enough to make the algorithm create more or less Gaussian components, which has a huge impact on the final model, like a butterfly effect. Moreover, the excessive creation of components can cause overfitting, as well as catastrophic forgetting. New components can have large overlapping regions with old components, interfering with what has been previously learned. Thus, we propose that, in future works, this issue must be addressed with a certain urgency, as this can deter practical use of the IGMN (the same issue was also found by [35] and explored by [28]).

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[5] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *arXiv preprint arXiv:1603.00748*, 2016.

[9] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[10] A. Aubret, L. Matignon, and S. Hassas, "A survey on intrinsic motivation in reinforcement learning," *arXiv preprint arXiv:1908.06976*, 2019.

[11] M. Heinen, P. Engel, and R. Pinto, "IGMN: An Incremental Gaussian Mixture Network that Learns Instantaneously from Data Flows," in *Proceedings of the IX ENIA - Brazilian Meeting on Artificial Intelligence*, Natal (RN), July 2011.

[12] M. R. Heinen, P. M. Engel, and R. C. Pinto, "Applying the incremental gaussian neural network to concept formation and robotic tasks," *Proc. 10th Brazilian Congr. Computational Intelligence (CBIC). Fortaleza, CE, Brazil (Nov 2011)*, 2011.

[15] A. Raffin, A. Hill, K. R. Traoré, T. Lesort, N. Díaz-Rodríguez, and D. Filliat, "Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics," *arXiv preprint arXiv:1901.08651*, 2019.

[13] ——, "Using a gaussian mixture neural network for incremental learning and robotics," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.

[14] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "Darla: Improving zero-shot transfer in reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1480–1490.

[16] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis, "Model-free episodic control," *arXiv preprint arXiv:1606.04460*, 2016.

[17] A. Agostini and E. Celaya, "Reinforcement learning with a gaussian mixture model," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[18] R. C. Pinto and P. M. Engel, "A fast incremental gaussian mixture model," *PloS one*, vol. 10, no. 10, p. e0139931, 2015.

[19] R. Pinto and P. Engel, "Scalable and incremental learning of gaussian mixture models," *arXiv preprint arXiv:1701.03940*, 2017.

[20] R. Dearden, N. Friedman, and S. Russell, "Bayesian q-learning," in *AAAI/IAAI*, 1998, pp. 761–768.

[21] M. Heinen, A. Bazzan, and P. Engel, "Dealing with continuous-state reinforcement learning for intelligent control of traffic signals," in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE, 2011, pp. 890–895.

[22] A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell, "Neural episodic control," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2827–2836.

[23] A. Agostinelli, K. Arulkumaran, M. Sarrico, P. Richemond, and A. A. Bharath, "Memory-efficient episodic control reinforcement learning with dynamic online k-means," 2019.

[24] R. Pinto, "Model-free episodic control with state aggregation," *arXiv preprint arXiv:2008.09685*, 2020.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2008.

[27] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on signal processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[28] J. C. Chamby-Diaz, M. Recamonde-Mendoza, A. L. Bazzan, and R. Grunitzki, "Adaptive incremental gaussian mixture network for non-stationary data stream classification," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[29] R. C. Pinto, "Continuous reinforcement learning with incremental gaussian mixture models," PhD dissertation, Universidade Federal do Rio Grande do Sul, 2017.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[31] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *Journal of Artificial Intelligence Research (JAIR)*, vol. 47, pp. 253–279, 2013.

[32] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477*, 2015.

[33] P. Engel and M. Heinen, "Concept Formation Using Incremental Gaussian Mixture Models," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 128–135, 2010.

[34] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998, vol. 1.

[35] D. Koert, S. Trick, M. Ewerton, M. Lutter, and J. Peters, "Online learning of an open-ended skill library for collaborative tasks," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 1–9.