2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

NEVE++: A Neuro-Evolutionary Unlimited Ensemble for Adaptive Learning

Tatiana Escovedo, André Abs da Cruz, Adriano Koshiyama, Rubens Melo, Marley Vellasco

Abstract— **In our previous works [1, 2], we proposed NEVE, a model that uses a weighted ensemble of neural network classifiers for adaptive learning, trained by means of a quantum-inspired evolutionary algorithm (QIEA). We showed that the neuro-evolutionary classifiers were able to learn the dataset and to quickly respond to any drifts on the underlying data. Now, we are particularly interested on analyzing the influence of an unlimited ensemble, instead of the limited ensemble from NEVE. For that, we modified NEVE to work with unlimited ensembles, and we call this new algorithm NEVE++. To verity how the unlimited ensemble influences the results, we used four different datasets with concept drift in order to compare the accuracy of NEVE and NEVE++, using two other existing algorithms as reference.**

Keywords— **Concept Drift, Adaptive Learning, Nonstationary Environments, Neuro-Evolutionary Ensemble, Quantum-Inspired Evolution.**

I. Introduction

REAL world concepts are often not stable: they change with time. Typical examples of scenarios where these changes are occurring are problems involving rules for climate prediction, detection of spam emails and customer preference. Just as the concepts, data distribution may change as well. The problem that occurs with learning algorithms that deal with these scenarios is that, usually, any of these changes make the model that was built based on old data inconsistent with the new data, resulting necessary to change the model accordingly so that learning is not impaired. This problem of change in concepts or distribution of data is known as concept drift and is a complication for a model in the task of learning from data. Specific strategies are needed, different from the techniques traditionally used in which arriving data samples are treated as equal contributors to the final concept [3].

These concept changes, in turn, may be small fluctuations in the underlying probability distributions, stable, random or systematic trends, rapid replacement of a classification task, among others. A classifier, be it individual or an ensemble must be equipped with some mechanism to adapt to changes in the environment [7].Therefore, the ability for a classifier to learn from incrementally updated data drawn from a nonstationary environment poses a challenge to the field of computational intelligence. Moreover, the use of neural networks as classifiers makes the problem even harder, as neural networks are usually seen as tools that must be retrained with the whole set of instances learned so far when a new chunk of data becomes available.

In order to cope with that sort of problem, a classifier must, ideally, be able to [5]:

- Track and detect any sort of changes on the underlying data distribution;
- Learn with new data without the need to present the whole data set again for the classifier;
- Adjust its own parameters in order to address the detected changes on data;
- Forget what has been learned when that knowledge is no longer useful for classifying new instances.

All those abilities try, in a way or another, to address a phenomenon called "concept drift" [3, 6]. This phenomenon defines data sets which suffer changes over time, like, for example, when the relevant variables change or either mean or variance of the time series is changing. Most work in the field of learning in non-stationary environments was published in the last decade and it is observed that until now there is lack of standard terminology. The use of different terms by the authors hinders comparison of proposals and studies in the area. The article [7] proposes a unification and standardization of this nomenclature and proposes the term *Dataset shift* to represent the general problem that occurs when the test data (not yet viewed) experience a phenomenon that leads to a change i) in the distribution of a single feature, ii) a combination of features or iii) in the boundaries of classes. As a result, the common assumption that training data and test follow the same distributions is often violated in real applications and scenarios. The authors also propose the following terms: *Covariate shift*, for changes in the distribution of the input variables x; *Prior probability shift*, for changes in the distribution of class variables y and *Concept Shift*, when the relationship between the inputs and class variables changes. Although we believe the approach presented interesting, in this study we chose to use the term *Concept drift*, first, because it is the term most used in the literature and secondly because we are not currently interested in analyzing the influence that each possible type of change would have on the final result of our model.

Many approaches have been devised in order to accomplish some or all of the abilities mentioned above. The

This work was partially funded by FAPERJ (Fundação de Amparo à Pesquisa do Rio de Janeiro) and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

All authors are with the Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rua Marquês de São Vicente, 225, Gávea - Rio de Janeiro, RJ - Brasil - 22451-900. (corresponding author phone: +55(21)9490-6545; e-mail: [tatiana, andrev, marley, adriano]@ ele.puc-rio.br, rubens@inf.puc-rio.br).

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

simplest one consists in using a sliding window on incoming data and training the classifier with the data delimited by that window [8]. Another approach consists in detecting drifts and then making adjustments to the classifier according to the drift.

A more successful approach consists in using an ensemble of classifiers. This kind of approach uses a group of different classifiers in order to be able to track changes on the environment. Several different models of ensembles have been proposed on the literature [4, 9, 10]:

- Ensembles that create new classifiers to each new chunk of data and weight classifiers according to their accuracy on recent data;
- Unweighted ensembles which can cope with new data that belongs to a concept different from the most recent training data;
- Ensembles that are able to discard classifiers as they become inaccurate or when a concept drift is detected.

Most models using weighted ensembles determine the weights for each classifier using some sort of heuristics related to the amount of mistakes the classifier does when working with the most recent data [6]. Although in principle any classifier can be used to build the ensembles, the ones which are most commonly used are decision trees, neural networks and naive Bayes [8].

This work represents an extension of [1, 2] where we presented an approach based on neural networks which are trained by means of a quantum-inspired evolutionary algorithm. Quantum-inspired evolutionary algorithms [12-16] are a class of estimation of distribution algorithms which present, for several benchmarks, a better performance for combinatorial and numerical optimization when compared to their canonical genetic algorithm counterparts. The quantum-inspired evolutionary algorithm for numerical optimization (QIEA-R), to be presented in section 3, has shown good performance when used to train a neural network for time series forecasting and control problems. Training a neural network by using an evolutionary algorithm can be beneficial, especially on reinforcement learning problems, when generating instances of input-output data is not simple or even possible. Moreover, by using an evolutionary algorithm for training a neural network, one can possibly be able to train complex architectures like networks with non-continuous activation functions and recurrent neural networks in a straightforward way and even to define the neural network's topology during the training [13, 17].

We also use the QIEA-R to determine the voting weights for each classifier that is part of the ensemble. Every time a new chunk of data arrives, a new classifier is trained on this new data set and all the weights are optimized in order for the ensemble to improve its performance on classifying this new set of data.

The main difference between NEVE (Neuro-Evolutionary Ensemble) [1, 2] and the model presented here is that now, we are particularly interested on investigating the impact of the ensemble size, and hence, we modified NEVE to work with unlimited ensembles, and we call this new model NEVE++. To evaluate the impact of this modification at the algorithm accuracy, we used four different datasets to execute several experiments, in order to compare NEVE++ and NEVE accuracies, using the existing algorithms Learn++.NSE [9] and Diversity for Dealing with Drifts (DDD) [18] only as reference values, not intending to specifically compare the algorithms.

This paper is organized in four additional sections. Section 2 presents some theoretical concepts related to concept drift and some existing approaches, including the Learn++.NSE algorithm and DDD, which are our basis of comparison in this work. Section 3 details the proposed model and Section 4 presents and discusses the results of the experiments. Finally, section 5 concludes this paper and presents some possible future works.

II. CONCEPT DRIFT

A. Definitions

The term concept drift can be defined informally as a change in the concept definition over time and, hence, change in its distribution. An environment from which this kind of data is obtained is considered a nonstationary environment.

Concept drift can also be defined as an obstacle caused by insufficient, unknown or unobservable features in a dataset, which happens in many real problems. These problems usually depend on a context that is not explicitly stated in the predicted features. This scenario is known as hidden context, and a typical example is the climatic prediction rules, that can vary radically according to the season of the year. Another example would be the analysis of consumption patterns that can vary in time, according to the month, availability of alternative products, inflation rate, etc. [3]. Analyzing the problem with the benefit of this hidden context would help solve the nonstationarity problem.

A practical example of concept drift mentioned in [10] is detecting and filtering out spam e-mails. The description of the two classes "spam" and "non-spam" may vary in time. They are user specific, and user preferences are also varying over time. Moreover, the variables used at time t to classify spam may be irrelevant at $t+k$. In this way, the classifier must deal with the "spammers", who will keep creating new forms to trick the classifier into labeling a spam as a legitimate e-mail.

B. Related Work

Algorithms designed for concept drift can be characterized in several ways. Based on [4, 9, 10], we propose a possible classification, as follows:

Active x Passive

- **Active:** Uses some drift detection mechanism, learning only when the drift is detected)
- **Passive** (Assume possibly ongoing drift and continuously update the model with each new data(set). If change has occurred, it is learned, else, the existing knowledge is reinforced.)

Online x Batch

- **Online:** Learn one instance at a time. They have better plasticity but poorer stability properties. They also tend to be more sensitive to noise as well as to the order in which the data are presented.
- **Batch:** Requires blocks of instances to learn. They benefit from the availability of larger amounts of data, have better stability properties, but can be ineffective if the batch size is too small, or if data from multiple environments are present in the same batch. Typically use some form of windowing to control the batch size.

Single Classifier x Ensemble

- **Single Classifier:** Uses only one classifier.
- **Ensemble:** Combines multiple classifiers.

The ensemble-based approaches that combine multiple classifiers constitute a new breed of nonstationary learning (NSL) algorithms. These algorithms tend to be more accurate, more flexible and sometimes more efficient than single classifiers [10]. Most of all uses some voting method, yet there is no agreement in the literature about the best type to be used.

Several ensemble approaches were already proposed in the literature, such as the Street's Streaming Ensemble Algorithm (SEA) [19], the Chen and He's Recursive Ensemble Approach (REA) [20], the Tsymbal's Dynamic Integration [21], the Kolter and Maloof's online algorithm Dynamic Weighted Majority (DWM) [22]. In this work, we used Learn++.NSE [9] and Diversity for Dealing with Drifts (DDD) [18] to compare the results reached by our proposed model. These algorithms will be presented briefly as follows.

1) Learn++.NSE

Developed based on the guidelines for building learning algorithms in nonstationary environments, previously presented in this section, Learn++.NSE [9] is an ensemble-based batch learning algorithm that uses weighted majority voting, where the weights are dynamically updated with respect to the classifiers' time-adjusted errors on current and past environments. The algorithm uses a passive drift detection mechanism, and uses only current data for training. It can handle a variety of non-stationary environments, including sudden concept change, or drift that is slow or fast, gradual or abrupt, cyclical, or even variable rate drift. It is also one of the few algorithms that can handle concept addition (new class) or deletion of an existing class.

The algorithm assumes that at each step may or may not have occurred change in environment and, if occurred, the rate of change is unknown and it is assumed that it is not constant. It is also assumed that all previously seen data (relevant or not for the learning) is not accessible or it is not possible to access it, meaning that the algorithm works incrementally. All relevant information about the previous data must be stored in the parameters of the previously generated classifiers. Depending on the nature of the change, the algorithm retains, builds or temporarily discards knowledge, so that new data can be categorized.

The knowledge base is initialized by creating a single classifier in the first data block available. Once prior knowledge is available, the current ensemble (knowledge base) is evaluated by current data: the algorithm identifies which new samples were not recognized by the existing knowledge base and this is updated by adding a new classifier trained on current training data. Each classifier (including the recently created) is evaluated on the training data. As previously unknown data have been identified, the penalty for misclassifying is considered in the error calculation. This way, more credit is given to the classifiers capable of identify previously unknown instances, while classifiers that misclassify previously known data are penalized. Then, classifier error is weighted considering the time: recent competence is taken more into account when categorizing knowledge. After that, the voting weights are determined: if knowledge of a classifier is not compatible with the current environment, it receives little or no weight and is temporarily removed from the knowledge base. It is not discarded: if its knowledge becomes relevant again, it will receive higher voting weights. The final decision is taken on with the weighted majority vote of the current ensemble members.

2) Diversity for Dealing with Drifts (DDD)

DDD [18] is an online ensemble approach that operates in 2 modes: prior to drift detection and after drift detection. It uses a drift detection method that detect drifts the earliest possible and it is designed to be robust to false alarms.

Before a drift is detected, the learning system is composed of two ensembles: an ensemble with lower diversity and an ensemble with higher diversity. Both ensembles are trained with incoming examples, but only the low diversity ensemble is used for system predictions, because the high diversity ensemble is likely to be less accurate on the new concept. DDD assumes that, if there is no convergence of the underlying distributions to a stable concept, new drift detections will occur, triggering the mode after drift detection. DDD then allows the use of the high diversity ensemble in the form of an old high diversity ensemble.

After a drift is detected, new low diversity and high diversity ensembles are created. The ensembles corresponding to the low and high diversity ensembles before the drift detection are kept and denominated old low and old high diversity ensembles. The old high diversity ensemble starts to learn with low diversity in order to improve its convergence to the new concept.

Both the old and the new ensembles perform learning and the system predictions are determined by the weighted majority vote of the output of the old high diversity, the new low diversity and the old low diversity ensemble. During the mode after drift detection, the new low diversity ensemble is monitored by the drift detection method. If two consecutive drift detections happen and there is no shift back to the mode prior to drift detection between them, the old low diversity ensemble after the second drift detection can be either the same as the old high diversity learning with low diversity

after the first drift detection or the ensemble corresponding to the new low diversity after the first drift detection, depending on which of them is the most accurate.

All the four ensembles are maintained in the system until either some conditions are satisfied, then, the system returns to the mode prior to drift detection. When returning to the mode prior to drift, either the old high diversity or the new low diversity ensemble becomes the low diversity ensemble used in the mode prior to drift detection, depending on which of them is the most accurate.

III. The Proposed Model

A. The Quantum-Inspired Neuro-Evolutionary Model

Neuro-evolution is a form of machine learning that uses evolutionary algorithms to train artificial neural networks. This kind of model is particularly interesting for reinforcement learning problems, where the availability of input-output pairs is often difficult or impossible to obtain and the assessment of how good the network performs is made by directly measuring how well it completes a predefined task. As training the weights in a neural network is a non-linear global optimization problem, it is possible to minimize the error function by means of using an evolutionary algorithm approach.

The quantum-inspired evolutionary algorithm (QIEA) is a class of "estimation of distribution algorithm" (EDA) that has a fast convergence and, usually, provides a better solution, with less evaluations than the traditional genetic algorithms [10, 11]. In this model, quantum-inspired genes are represented by probability density functions (PDF) which are used to generate classical individuals through an observation operator. After being observed, the classical individuals are evaluated, as in traditional genetic algorithms, and, by means of using fitness information, a set of quantum-inspired operators are applied to the quantum individuals, in order to update the information they hold in such a way that on the next generations, better individuals will have a better chance to be selected. Further details on how this global optimization method works can be found in [12-16].

Based on this algorithm, the proposed quantum-inspired neuro-evolutionary model consists in a neural network (a multilayer perceptron (MLP)) and a population of individuals, each of them encoding a different configuration of weights and biases for the neural network. If the neural network has n_i inputs, n_h hidden processors and n_o outputs, then the total number of weights and biases that must be encoded by the genes in the individuals is given by

$$t_p = n_i * n_h + n_h + n_h * n_o + n_o \quad (1)$$

which considers the connections between the inputs and the hidden processors, the connections between the hidden processors and the output processors and the biases for the hidden and output processors.

The training process occurs by building one MLP for each classical individual using the genes from this individual as weights and biases. After that, the full training data set (or the set of tasks to be performed) is presented to the MLP and the average error regarding the data set is calculated for each MLP. This average error is used as the fitness for each individual associated to that MLP, which allows the evolutionary algorithm to adjust itself and move on to the next generation, when the whole process will be repeated until a stop condition is reached. The individual is shown in figure 1. The whole process of training the neural network by using the quantum-inspired evolutionary algorithm can be summarized as shown in figure 2.

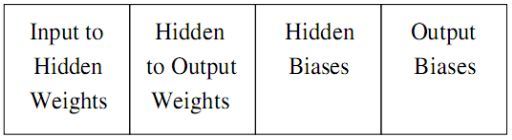

Fig. 1. The QIEA-R individual structure.

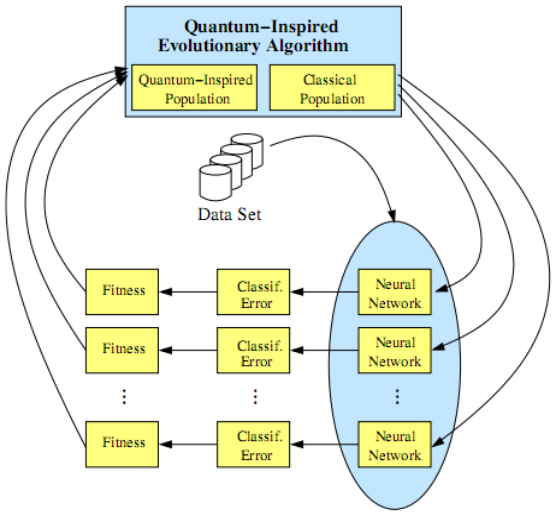

Fig. 2. The quantum-inspired neuro-evolutionary model.

B. NEVE++: The Neuro-Evolutionary Unlimited Ensemble

To some applications, such as those that use data streams, the strategy of using simpler models is most appropriate because there may not be time to run and update an ensemble. However, when time is not a major concern, yet the problem requires high accuracy, an ensemble is the natural solution. The greatest potential of this strategy for detecting drifts is the ability of using different forms of detection and different sources of information to deal with the various types of change [4].

One of the biggest problems in using a single classifier (a neural network, for example) to address concept drift problems is that when the classifier learns a dataset and then we need it to learn a new one, the classifier must be retrained with all data, or else it will "forget" everything already learned. Otherwise, using the ensemble, there is no need to retrain it again, because it can "retain" the previous knowledge and still learn new data.

Hence, in order to be able to learn as new chunks of data arrive, we implemented an ensemble with neural networks that are trained by an evolutionary algorithm, presented in

section 2.B. This approach makes the ensemble useful for online reinforcement learning, for example. The algorithm works as shown in figure 3 and each step is described in detail on the next paragraphs.

On step 1 we create the empty ensemble with a predefined size equal to *s*. When the first chunk of data is received, a neural network is trained using the QIEA-R until a stop condition is reached (for example, the number of evolutionary generations or an error threshold). As the number of classifiers in the ensemble is smaller than *s* considering $s=\infty$, we simply add this new classifier to the ensemble. This gives the ensemble the ability to learn the new chunk of data without having to parse old data. This is the main difference between our previous algorithm NEVE and the current algorithm NEVE++. In NEVE, if the ensemble was already full, we needed to evaluate each classifier on the new data set and remove the one with the highest error rate (including the new one, which means the new classifier will only become part of the ensemble if its error rate is smaller than the error rate of one of the classifiers already in the ensemble). Since NEVE++ continuously adds classifiers, one may be concerned about proliferation of classifiers. We decided to try this approach instead of using a fixed ensemble size and removing additional classifiers based on their error (as we did at [2]). At [23], the authors showed that performance benefits of retaining the ensemble far outweighs the additional and modest computational and memory costs and they do not recommend the fixed size approach because , it reduces the ability of the algorithm to remember recurring environments as well as its stability during stationary periods.

1. Create an empty ensemble P
2. Define the ensemble size ∞
3. For each chunk of data D_i, $i = 1, 2, 3, ..., m$ do
3.1. Train the classifier using the QIEA-R and a MLP and calculate its error E' over the data chunk
3.2. Add the new classifier to the ensemble
3.3. Evolve the voting weights w_j for each classifier in the ensemble using the last chunk of data D_i

Fig. 3. The neuro-evolutionary unlimited ensemble training algorithm.

Finally, we use the QIEA-R to evolve a voting weight for each classifier. Optimizing the weights allows the ensemble to quickly adapt to sudden changes on the data, by giving higher weights to classifiers better adapted to the current concepts governing the data. The chromosome that encodes the weights has one gene for each voting weight, and the population is evolved using the classification error as the fitness function. It is important to notice that when the first *s-1* data chunks are received, the ensemble size is smaller than its final size and thus, the chromosome size is also smaller. From the s data chunk on, the chromosome size will remain constant and will be equal to *s*.

In this work we used only binary classifiers but there is no loss of generality and the algorithm can be used with any number of classes. For the binary classifier, we discretize the neural network's output as "1" or "-1" and the voting process for each instance of data is made by summing the NN's output multiplied by its voting weight. In other words, the ensemble's output for one instance *k* from the i-th data chunk is given by:

$$P(D_{ik}) = \sum_{j=0}^{s} w_j c_j(D_{ik}) \quad (2)$$

where $P(D_{ik})$ is the ensemble's output for the data instance D_{ik}, w_j is the weight of the j-th classifier and $c_j(D_{ik})$ is the output of the j-th classifier for that data instance. If $P(D_{ik}) < 0$, we assume the ensemble's output is "-1". If $P(D_{ik}) > 0$, we assume the ensemble's output is "1". If $P(D_{ik}) = 0$, we choose a class randomly.

Thus, the main difference between our model, the Learn++.NSE and the DDD algorithms is that we use a neuro-evolutionary approach, based on a quantum-inspired algorithm to train the neural networks and to determine the voting weights for each member of the ensemble. All algorithms use an ensemble strategy. NEVE++ and Learn++.NSE algorithms use passive and batch approaches; DDD uses active and online approaches, according to the proposed classification presented.

IV. Experimental Results

In order to check the ability of our model on learning data sets with concept drifts and compare its accuracy with its previously version (NEVE, which uses a fixed ensemble size), we used four different data sets (SEA Concepts and Nebraska, also used at [10]; Circle and Line, also used at [18, 24]) upon which we performed several simulations in different scenarios. All experiments begin at $t=0$ and end at an arbitrary time $t=1$. Meanwhile, T consecutive data blocks are presented for training, each one taken from a possible drift scenario, where the rate or nature is unknown. The value T determines the number of time steps (or snapshots) taken from the data during the drift period.

On each experiment, we used a fixed topology for the neural networks consisting of 2 inputs for Circle and Line dataset, 3 inputs for SEA Concepts dataset and 8 inputs for Nebraska dataset, representing the input variables for each dataset. In all datasets, we used 1 output, and we used 5 neurons for the hidden layer, because this was the best value found by some previously analysis [1, 2]. Each neuron has a hyperbolic tangent activation function and, as mentioned before, the output is discretized as "-1" or "1" if the output of the neuron is negative or positive, respectively. The evolutionary algorithm trains each neural network for 100 generations. The quantum population has 10 individuals and the classical population 20. The crossover rate is 0:9 (refer to [12, 13] for details on the parameters). The same parameters are used for evolving the weights for the classifiers.

The neural network weights and biases and the ensemble weights are allowed to vary between -1 and 1 as those values are the ones who have given the best results on some pre-evaluations we have made. The hidden layer neuron number and the ensemble size for NEVE, in each experiment, were given using the results of the best configuration found by a

previously analysis using different values, made in [1].

We made, for each experiment, statistical comparisons between the results found by Learn++.NSE, DDD, NEVE and NEVE++ algorithms. These were based on the correct classification performance throughout the test phase for each method. The results of Learn++.NSE can be found at [9] and the results of DDD, at [18]. For each dataset used, due to the stochastic optimization algorithm used to train NEVE and NEVE++:

- We made 10 runs of NEVE++ and calculated the mean error e1;
- We made 10 runs of NEVE using ensemble size = 5 and calculated the mean error e2;
- We made 10 runs of NEVE using ensemble size = 10 and calculated the mean error e3;
- We compared e1, e2 and e2 with the results of Learn++ (for SEA and Nebraska datasets) and with the results of DDD (for Circle and Line datasets).

As mentioned above, we are particularly interested on investigating the impact of the ensemble size, and hence, we modified NEVE to work with unlimited ensembles, creating NEVE++. This way, our main interest is to observe how this modification affects the accuracy of the algorithm. We just used the results of Learn++.NSE [9] and Diversity for Dealing with Drifts (DDD) [18] as reference values, not intending to specifically compare the algorithms.

Based on these runs, we calculate some statistical parameters (mean, standard deviation, etc.) that were used to compute the Welch t-test [25] to evaluate which algorithm had, in average, the best performance in test phase. The normality assumption necessary for Welch t-test was verified using Shapiro-Wilk test [26]. All the statistical analysis were conducted in R statistical package [27].

A. SEA Concepts

The SEA Concepts was developed by Street [19] and has been used by several algorithms as a standard test for concept change. The dataset, available at [28], is characterized by extended periods without any drift with occasional sharp changes in the class boundary, i.e., sudden drift or concept change. The dataset consists of 50000 random points in a three-dimensional feature space. The features are in the [0; 10] domain but only two of the three features are relevant to determine the output class. These points are then divided into four blocks, with different concepts. Class labels are assigned based on the sum of the relevant features, and are differentiated by comparing this sum to a threshold that separates a 2-D hyper-plane: an instance is assigned to class 1 if the sum of its (relevant) features ($f_1 + f_2$) fall below the threshold, and assigned to class 2, otherwise. At regular intervals, the threshold is changed with increasing severity (8→9→7.5→9.5), creating an abrupt shift in the class boundary.

Aiming to enable a better comparison with the results of the algorithm Learn++. NSE, detailed in section 2, we used 200 blocks of size 250 to evaluate the algorithm in the test phase. Then, NEVE++, NEVE with ensemble size = 5, NEVE with ensemble size = 10 and Learn++.NSE results were displayed in Table 1.

TABLE I. RESULTS OF THE SEA EXPERIMENTS.

Algorithm	Mean	Standard Deviation
NEVE ++	2.04%	0.10%
NEVE(5)	1.65%	0.14%
NEVE(10)	1.69%	0.13%
Learn++.NSE (SVM)	1.79%	0.20%

As can be seen, NEVE++ performed a little worse than NEVE(5) and NEVE(10), indicating that for this dataset, the use of an unlimited ensemble is not the best strategy. Despite the better accuracy of Learn++.NSE for this dataset, numerically the difference is little (less than 0.5%), but when performing a statistical comparison we can see that Learn++.NSE performed in average significantly better than NEVE++ (t_{crit} = 4.85, *p-value* < 0.01).Further details of Learn++.NSE results can be found in [9].

Next subsection exhibits the second experiment, based on Nebraska Weather data.

B. Nebraska Weather Prediction Data

This dataset, also available at [28], presents a compilation of weather measurements from over 9000 weather stations worldwide by the U.S. National Oceanic and Atmospheric Administration since 1930s, providing a wide scope of weather trends. Daily measurements include a variety of features (temperature, pressure, wind speed, etc.) and indicators for precipitation and other weather-related events. As a meaningful real world dataset, we chose the Offutt Air Force Base in Bellevue, Nebraska, for this experiment due to its extensive range of 50 years (1949–1999) and diverse weather patterns, making it a long-term precipitation classification/prediction drift problem.

Class labels are based on the binary indicator(s) provided for each daily reading of rain: 31% positive (rain) and 69% negative (no rain). Each training batch consisted of 30 samples (days), with corresponding test data selected as the subsequent 30 days. Thus, the learner is asked to predict the next 30 days' forecast, which becomes the training data in the next batch. The dataset included 583 consecutive "30-day" time steps covering 50 years.

Aiming to enable a better comparison with the results of the algorithm Learn++. NSE, we performed similarly to that used in [9]: we used 400 blocks of size 30 to evaluate the algorithm in the test phase. Then, NEVE++, NEVE with ensemble size = 5, NEVE with ensemble size = 10 and Learn++.NSE results were displayed in Table 2.

TABLE II. RESULTS OF THE NEBRASKA EXPERIMENTS.

Algorithm	Mean	Standard Deviation
NEVE++	29.51%	0.53%
NEVE(5)	31.85%	0.32%
NEVE(10)	31.44%	0.61%
Learn++.NSE (SVM)	21.20%	1.00%

As can be seen, the mean error rate of Learn++.NSE is the lowest, but comparing NEVE to NEVE++, we noticed that the mean error rate of NEVE++ is substantially lower than NEVE(5) and NEVE(10) (*p-value* < 0.01, t_{crit} = 10.7862 and t_{crit} = 4.85, respectively), indicating that for this dataset, the unlimited ensemble is a good strategy. Further details of Learn++.NSE results can be found in [9]. We see that the unlimited ensemble from NEVE++ produced better results compared to the previous version of our algorithm NEVE, but Learn++.NSE's results show that we have some improvement opportunities in our algorithm aiming to reach better results.

Next subsection exhibits the third experiment, based on Circle and Line datasets.

C. Circle and Line Datasets

In order to analyze the effect of diversity in the presence of concept drift, Minku [24] developed a data sets generator to create datasets with different types of drift for four problems, as figure 4 shows:

- Circle $(x-a)^2+(y-b)^2 \leq/> r^2$
- Sine $y \leq/> a\,sin(bx+c)+d$
- Moving hyperplane $y \leq/> -a_0+\sum_{i=1}^{d} a_i x_i$
- Boolean
  $y = (color\ =_1/\neq_1\ a\ \vee_1/\wedge_1$
  $shape\ =_2/\neq_2\ b)\ \vee_2/\wedge_2$
  $size\ =_3/\neq_3\ c$

where $a, b, c, d, r, a_i, =/\neq, \vee/\wedge$ and $\leq/>$ can assume different values to define different concepts.

Fig. 4. Details of the artificial datasets [24].

The dataset is available at [29]. The examples generated contain x/xi and y as the input attributes and the concept (which can assume value 0 or 1) as the output attribute. The range of x or x_i and y was [0,1]. Eight irrelevant attributes and 10% class noise were introduced in the plane data sets. Each data set contains 1 drift and different drifts were simulated by varying among 3 amounts of severity, generating 3 different datasets for each problem. In this study, we decided to use only circle and line (moving hyperplane with d=1) datasets, detailed as follows:

- Circle: a = b = 5; r = 0.2 → 0.3 (severity 1); r = 0.2 → 0.4 (severity 2); r = 0.2 → 0.5 (severity 3)
- Line: a1 = 0.1; a0=−0.4 → −0.55 (severity 1); a0=−0.25 → −0.7 (severity 2); a0=−0.1 → −0.8 (severity 3)

Because the DDD algorithm works very different from NEVE++, as we already mentioned, it is hard to reproduce the same settings aiming to make comparisons, but we still decided to use DDD results only as a referential value just to check if our accuracy is satisfactory. Then, NEVE++, NEVE with ensemble size = 5, NEVE with ensemble size = 10 and DDD results were displayed in Table 3 (circle datasets) and 4 (line datasets).

TABLE III. RESULTS OF THE CIRCLE EXPERIMENTS

	Severity 1		Severity 2		Severity 3	
Algorithm	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
NEVE++	15.20%	1.00%	19.24%	1.96%	16.08%	1.89%
NEVE(5)	15.37%	2.88%	19.41%	3.54%	17.89%	0.65%
NEVE(10)	16.93%	1.90%	14.74%	1.02%	16.25%	1.49%
DDD	7.39%	0.96%	8.83%	0.91%	10.06%	1.01%

TABLE IV. RESULTS OF THE LINE EXPERIMENTS

	Severity 1		Severity 2		Severity 3	
Algorithm	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
NEVE++	1.85%	0.52%	2.54%	0.55%	3.65%	0.56%
NEVE(5)	1.22%	0.54%	2.30%	0.47%	3.55%	0.36%
NEVE(10)	1.71%	0.52%	2.29%	0.41%	3.44%	0.36%
DDD	4.83%	0.59%	5.99%	0.51%	6.73%	0.72%

Comparing the mean error rate of NEVE++, NEVE(5) and NEVE(10) for all 3 Circle datasets, the results are not the same. As can be seen, at Circle Severities 1 and 3 dataset, NEVE++ presented a better mean error rate, compared to NEVE(5) and NEVE(10). However, at Circle Severity 2, NEVE(10) had the best mean error rate, followed by NEVE++ and then by NEVE(5). For severity 1 and 3 this difference was not significant (*p-value* > 0.5), however for severity 2 mean error rate of NEVE(10) is substantially lower than NEVE(5) and NEVE++ approach (*p-value* < 0.01).

Although the difference among NEVE++, NEVE(5) and NEVE(10) mean error rates were small for all severities (less than 2%), the DDD algorithm always presented better accuracy results for this dataset. Nevertheless, the results indicated that the strategy of using an unlimited ensemble tends to be better than using a limited ensemble.

In the other hand, at Line dataset, NEVE and NEVE++ always presented better accuracy than DDD; the only experiment where NEVE++ was better than NEVE, although, was at severity 3, but the difference among NEVE and NEVE++ error rates were very small for all severities (less than 0.5%) and not statistically significant (*p-value* > 0.05), except in severity 1.

This section presented the results obtained in experiments taken with the NEVE++ algorithm. We detailed the results of four different datasets and then those results were compared with the results of other algorithms. The next section concludes this work.

V. CONCLUSIONS AND FUTURE WORKS

This paper presented a model that uses an unlimited ensemble of neural networks trained by a quantum-inspired evolutionary algorithm to learn datasets (possibly with

concept drifts) incrementally. We analyzed the ability of the model using four different datasets, using the algorithms Learn++.NSE and DDD as reference.

Although the NEVE++ algorithm has demonstrated a better performance when compared to NEVE in some datasets, these results are not conclusive because we also had worse results compared to NEVE, demonstrating that just for some situations and datasets, the unlimited ensemble strategy is better than the limited one. It is strongly recommended that further tests may be performed with different configurations to confirm whether or not the results presented here. It is also desirable to do a comparison with others algorithms in others datasets.

In the future, we intend to check the performance of NEVE and NEVE++ on other real data sets, although it is not easy to determine if a real world data set has any kind of significant changes on data. In any case, it is always possible to introduce these changes on any real data set, artificially. We also intend in the future to continue this work, analyzing other existing approaches, such as [22] and [30], and performing new experiments comparing our work with these and other algorithms.

We still need to do some changes in NEVE++ to verify if we can reach better results. For example, a possibility is to use the voting weights or some of the neural network weights (for instance, the weights from the input to the hidden neurons) to detect concept drifts and to be able to direct better the learning process. Another idea is to use the evolutionary process to evolve the voting weights for more generations if we detect a significant change on the underlying data. This might allow the ensemble to "react" faster to the concept drift. Another possibility is to use the QIEA to evolve and find the ideal number of neurons in the hidden layer for each member of the ensemble. In addition, we intend to perform a deep sensibility analysis for parameters that compose NEVE++ (QIEA algorithm and MLP), in order to better address the impact of each configuration and find an almost optimal setting.

REFERENCES

[1] T. Escovedo, A. V. Abs da Cruz, M. Vellasco and A. Koshiyama. "NEVE: A Neuro-Evolutionary Ensemble for Adaptive Learning." In Artificial Intelligence Applications and Innovations, pp. 636-645. Springer Berlin Heidelberg, 2013.

[2] T. Escovedo, A. V. Abs da Cruz, M. Vellasco and A. Koshiyama. "Using ensembles for adaptive learning: A comparative approach." In Neural Networks (IJCNN), The 2013 International Joint Conference on, pp. 1-7. IEEE, 2013.

[3] A. Tsymbal, "The problem of concept drift: Definitions and related work", Tech. Rep., 2004.

[4] L. I. Kuncheva, "Classifier ensemble for detecting concept change in streaming data: Overview and perspectives," in Proc. Eur. Conf. Artif. Intell, pp. 5–10, 2008.

[5] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data", Machine Learning, vol. 1, no. 3, pp. 317–354, 1986.

[6] M. T. Karnick, M. Ahiskali, M. Muhlbaier, and R. Polikar, "Learning concept drift in nonstationary environments using an ensemble of classifiers based approach," in IJCNN, pp. 3455–3462, 2008.

[7] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, e F. Herrera. "A unifying view on dataset shift in classification." Pattern Recognition 45, no. 1, pp. 521-530, 2012.

[8] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," In Proc. Of The 2001 Acm Sigkdd Intl. Conf. On Knowledge Discovery And Data Mining, pp. 97–106, 2001.

[9] Ryan Elwell, Robi Polikar: Incremental Learning of Concept drift in Nonstationary Environments. IEEE Transactions on Neural Networks 22(10): 1517-1531, 2011.

[10] L. I. Kuncheva, "Classifier ensemble for changing environments," in Multiple Classifier Systems, vol. 3077. New York: Springer-Verlag, 2004.

[11] N. C. Oza, "Online Ensemble Learning," Dissertation, University of California, Berkeley, 2001.

[12] A. V. Abs da Cruz, M. M. B. R. Vellasco, and M. A. C. Pacheco, "Quantum-inspired evolutionary algorithms for numerical optimization problems," in Proceedings of the IEEE World Conference in Computational Intelligence, 2006.

[13] A. V. Abs da Cruz, "Algoritmos evolutivos com inspiração quântica para otimização de problemas com representação numérica," Ph.D. dissertation, Pontifical Catholic University – Rio de Janeiro, 2007.

[14] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization", IEEE Trans. Evolutionary Computation, vol. 6, no. 6, pp. 580–593, 2002.

[15] K.-H. Han and J.-H. Kim, "On setting the parameters of qea for practical applications: Some guidelines based on empirical evidence," in GECCO, pp. 427–428, 2003.

[16] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithms with a new termination criterion, H_e gate, and two-phase scheme," IEEE Trans. Evolutionary Computation, vol. 8, no. 2, pp. 156–169, 2004.

[17] W. Lieffijn, "Heterogeneous neuro-evolutionary specialization of collective rover behaviors," 2008.

[18] L. L. Minku and X. Yao, "DDD: A New Ensemble Approach For Dealing With Concept Drift,", IEEE Transactions on Knowledge and Data Engineering, IEEE, v. 24, n. 4, p. 619-633, 2012.

[19] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., pp. 377–382, 2001

[20] S. Chen and H. He, "Toward incremental learning of nonstationary imbalanced data stream: A multiple selectively recursive approach," Evolv. Syst., vol. 2, no. 1, pp. 35–50, 2011.

[21] A. Tsymbal, M. Pechenizkiy, P. Cunningham and S. Puuronen, "Dynamic integration of classifiers for handling concept drift," Inf. Fus., vol. 9, no. 1, pp. 56–68, 2008.

[22] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," J. Mach. Learn. Res., vol. 8, pp. 2755–2790, 2007.

[23] R. Elwell and R. Polikar, "Incremental learning in nonstationary environments with controlled forgetting," in Proc. Int. Joint Conf. Neural Netw., Atlanta, GA, pp. 771–778, 2009.

[24] L. L. Minku, A. P. White and X. Yao, "The Impact of Diversity on On-line Ensemble Learning in the Presence of Concept Drift.", IEEE Transactions on Knowledge and Data Engineering, IEEE, v. 22, n. 5, p. 730-742, 2010.

[25] P. Dalgaard. "Introductory Statistics with R". New York: Springer-Verlag, 2002.

[26] P. Royston. "An extension of Shapiro and Wilk's W test for normality to large samples". Applied Statistics, vol. 31, pp. 115–124, 1982.

[27] R Development Core Team." R: A language and environment for statistical computing". R Foundation for Statistical Computing. Vienna, Austria. Donwload at: www.r-project.org, 2012.

[28] R. Polikar and R. Elwell. "Benchmark Datasets for Evaluating Concept drift/NSE Algorithms". At: http://users.rowan.edu/~polikar/research/NSE. Last access at December 2013.

[29] L. L. Minku, "Artificial Concept Drift Data Sets". At: http://www.cs.bham.ac.uk/~minkull/. Last access at January 2014.

[30] K. Jackowski. "Fixed-size ensemble classifier system evolutionarily adapted to a recurring context with an unlimited pool of classifiers." Pattern Analysis and Applications, 2013.