cPNN: Continuous Progressive Neural Networks for Evolving Streaming Time Series

Federico Giannini, Giacomo Ziffer, Emanuele Della Valle

DOI: https://doi.org/10.1007/978-3-031-33383-5_26

Originally published at:
Giannini, F., Ziffer, G., Della Valle, E. (2023). cPNN: Continuous Progressive Neural Networks for Evolving Streaming Time Series. In: Kashima, H., Ide, T., Peng, WC. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2023. Lecture Notes in Computer Science(), vol 13938.
DOI: https://doi.org/10.1007/978-3-031-33383-5_26
Publisher: **Springer, Cham**

cPNN: Continuous Progressive Neural Networks for Evolving Streaming Time Series

Federico Giannini[1][0000-0002-4210-6271], Giacomo Ziffer[1][0000-0002-2768-3580], and Emanuele Della Valle[1][0000-0002-5176-5885]

DEIB, Politecnico di Milano, Milano, Italy
{federico.giannini,giacomo.ziffer,emanuele.dellavalle}@polimi.it

Abstract. Dealing with an unbounded data stream involves overcoming the assumption that data is identically distributed and independent. A data stream can, in fact, exhibit temporal dependencies (i.e., be a time series), and data can change distribution over time (concept drift). The two problems are deeply discussed, and existing solutions address them separately: a joint solution is absent. In addition, learning multiple concepts implies remembering the past (a.k.a. avoiding catastrophic forgetting in Neural Networks' terminology). This work proposes Continuous Progressive Neural Networks (cPNN), a solution that tames concept drifts, handles temporal dependencies, and bypasses catastrophic forgetting. cPNN is a continuous version of Progressive Neural Networks, a methodology for remembering old concepts and transferring past knowledge to fit the new concepts quickly. We base our method on Recurrent Neural Networks and exploit the Stochastic Gradient Descent applied to data streams with temporal dependencies. Results of an ablation study show a quick adaptation of cPNN to new concepts and robustness to drifts.

Keywords: Data streams · Catastrophic forgetting · Concept drift

1 Introduction

In a context where data comes as an unbounded data stream and is continually evolving, we must overcome the central hypothesis of Machine Learning (ML): the assumption according to which data is independent and identically distributed (shortly, i.i.d). It does not hold for any data stream where data could suffer from changes in its distribution (the so-called "concept drift") and shows temporal dependencies. While the literature has deeply investigated the two situations separately, few works deal with the joint problem. The need to find a combined solution is, thus, increasingly emerging. We formalize the mentioned problem by calling it **Evolving Streaming Time Series (ESTS)**. "Evolving" indicates the possibility of concept drift, while "Streaming" refers to data points arriving continually from an unbounded data stream. We use "Time Series" to stress the presence of temporal dependencies. Working with concept drifts and multiple concepts makes it necessary to consider the well-known stability-plasticity dilemma [14], according to which too much plasticity results in forgetting past knowledge. This problem is known as **catastrophic forgetting (CF)** [10]. Too much stability leads, instead, to difficulties in learning new knowledge.

Among the models for dealing with time series, sequential models based on Recurrent Neural Networks (RNN) are widely used in the literature [7]. Applying Neural Networks (NN) to the streaming scenario allows it to exploit its learning algorithm's (Stochastic Gradient Descent, SGD) adaptability. In contrast, SGD can suffer when the new concept differs significantly from the previous one, and a NN forgets the last concept when it learns a new one. To resolve these issues, Progressive Neural Networks (PNN) [19] consist of NN architectures to jointly remember the previously learned knowledge and use transfer learning to recycle the knowledge gained from old concepts [16]. However, this methodology is not meant to deal with an ESTS.

Our work, thus, aims to investigate the following **research question**: *in the context of an Evolving Streaming Time Series, is there a solution to jointly manage concept drifts, temporal dependencies, and catastrophic forgetting?* In this paper, we positively answer this question by contributing **Continuous PNN (cPNN)**, a novel continuous version of PNNs that extends them to an ESTS scenario. We first propose a strategy to exploit SGD in a streaming scenario to tame temporal dependencies. Secondly, our approach utilizes PNN-based architectures to efficiently address both concept drifts and CF, using transfer learning to enable rapid adaptation to new concepts while maintaining the predictive ability of previously learned ones. A crucial feature of cPNN is that the architecture can be potentially applied to each type of RNN model. We conduct an ablation study on a binary classification problem during the experiment phase to test cPNN on synthetically generated data streams. We compare cPNN with two ablated architectures: cLSTM and mcLSTM. After a concept drift, cLSTM continues training on the new concept. It, thus, does not avoid CF and is not concept drift aware. mcLSTM avoids CF, but it does not use transfer learning. Temporal dependencies are tamed by using RNN models. Results show that cPNN performs better after concept drifts than ablated architectures.

The rest of the paper is organized as follows. Firstly, Section 2 analyzes the already present ideas in literature. Section 3 exposes our method and contributions. Then, Section 4 discusses the settings of our experiments, while Section 5 exhibits the results. Finally, Section 6 discusses conclusions and future works.

2 Related Works

Continual Learning (CL) thoroughly investigated methods to learn and avoid CF continually [12]. The Task Incremental Learning scenario assumes that data is split into batches of samples (named experiences) provided over time. Each of them represents a task. The data distribution and objective function are normally fixed within a task. In this paper, we refer to this scenario whenever we use CL.

In this context (shown in Fig. 1.a), **PNNs** [19] are NN architectures that use transfer learning to recycle knowledge gained from previous tasks. Furthermore, the parameters associated with the old tasks are frozen to avoid CF. PNNs, thus,

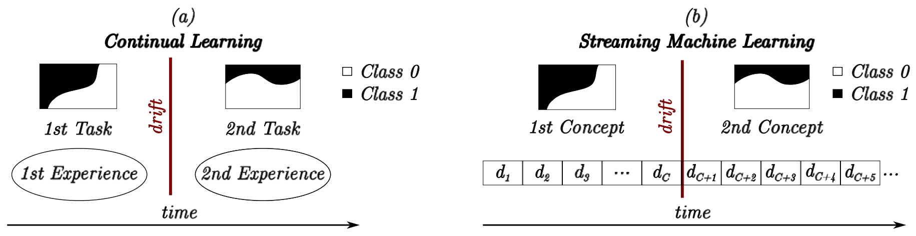

Fig. 1. Comparison of CL and SML scenarios.

can learn a new task while keeping the predictive ability on the earlier tasks. The architecture is built dynamically and starts with a single NN (named **column**). Equation 1 shows that, for each new task k, a column is added whose *i-th* layer receives the *(i-1)-th* layer's output $h_{i-1}^{(k)}$ of column k and the *(i-1)-th* layers' outputs $h_{i-1}^{(j)}$ of all the earlier columns. W_i and U_i are the weight matrices to be learned. U_i are called **lateral connections** and implement transfer learning.

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)})\right) \tag{1}$$

PNNs, on their own, do not tame temporal dependencies. To do so, they must use **RNNs** as columns, which operate on fixed-size sequences of items. RNNs recursively express the *i-th* hidden layer's output h_i as a function of the *i-th* item's features X_i and the output h_{i-1} of the *(i-1)-th* hidden layer. Due to the vanishing gradient, such an architecture cannot tame long temporal dependencies [7]. **Long short-term memory (LSTM)** [8] resolves this issue by memorizing only the helpful information and introducing the memory cell representing past cumulated knowledge. **Gated Incremental Memory (GIM)** [4] develops a recurrent version of PNNs using LSTM as columns. Column k receives lateral connections only from column $k-1$ to decrease the number of parameters. The *i-th* item's hidden layer $h_i^{(k)}$ of column k is computed as expressed by Equation 2. For each item i, its features X_i and the previous column's *i-th* hidden layer output are concatenated. Lateral connections are represented by the weights applied to the output of the previous column's hidden layer. The model's output is computed for each sequence element i by applying a further layer after h_i.

$$h_i^{(k)} = LSTM([X_i, h_i^{(k-1)}]) \tag{2}$$

The works mentioned above assume all data in each experience to be accessible at once. The specific paradigm called **Streaming Machine Learning (SML)** [3], instead, was introduced to learn continually from a data point (or mini-batch) at a time (see Fig. 1.b). **Concept drift**, that is a phenomenon in which the statistical properties of a target domain change over time in an arbitrary way [13], is a crucial issue that SML tames. We can distinguish two types of concept drift: virtual and real. It is easy to take them apart in the context of streaming classification. Virtual concept drifts do not affect the decision boundary, while real ones do. Additionally, in an abrupt drift, the new concept replaces the old one in a short period or in an exact instant, while in gradual and incremental drifts, the new concept gradually or incrementally replaces the old one. Finally, the concepts could reoccur over time. Concept drift detectors can detect all the mentioned types of concept drift [13].

Most SML methods assume that the data stream's points are independent. In the real world, this assumption is unrealistic since they can exhibit temporal dependencies. Despite many works raising the issue that ignoring this situation can cause problems in the learning and evaluation processes [18, 23, 24], taming of temporal dependencies in an evolving data stream is still an open issue.

3 Proposed Method

This work proposes cPNN, a novel methodology for applying NNs to perform binary classification of an ESTS' data points. In Section 3.1, we analyze SGD behavior on data streams containing concept drifts. Section 3.2 proposes a method to exploit SGD in an ESTS scenario. Finally, Section 3.3 presents cPNN.

3.1 Stochasting Gradient Descent for Evolving Data Streams

The SGD's iterative nature makes it possible to apply it on data streams by buffering the data points in fixed-size batches [7].[1] Fig. 2 illustrates this idea by analyzing a NN composed of a single linear neuron with two weights and no bias. Let's assume that the NN at d_{C1}, when a first abrupt drift occurs, has learned the decision boundary illustrated in Fig. 2.a. Notice that the second concept only marginally modifies the boundary between classes. Thus, SGD can quickly adapt to the drift since the minimum of the new concept's loss function is close to the previous one. On the contrary, the third concept swaps the classes when it occurs at d_{C2}. In this case, the new minimum is distant, and the SGD algorithm requires more iterations to reach it. Furthermore, the performance initially collapses since the starting configuration optimizes the inverted problem. In any case, when the model adapts to the new concept, it forgets the previous one since SGD has reached the new minimum. The more the new decision boundary changes, the lower the performance. Thus, a simple NN cannot deal with CF.

3.2 Stochasting Gradient Descent for Streaming Time Series

As already stressed, although the i.i.d. assumption is usually made for each concept, data can show time dependence that requires RNN models like LSTM. To ensure that SGD is an unbiased gradient, we cannot sample an entire i.i.d.

[1] See MOA's Perceptron application: `https://www.cs.waikato.ac.nz/~abifet/MOA/API/classmoa_1_1classifiers_1_1functions_1_1_perceptron.html`

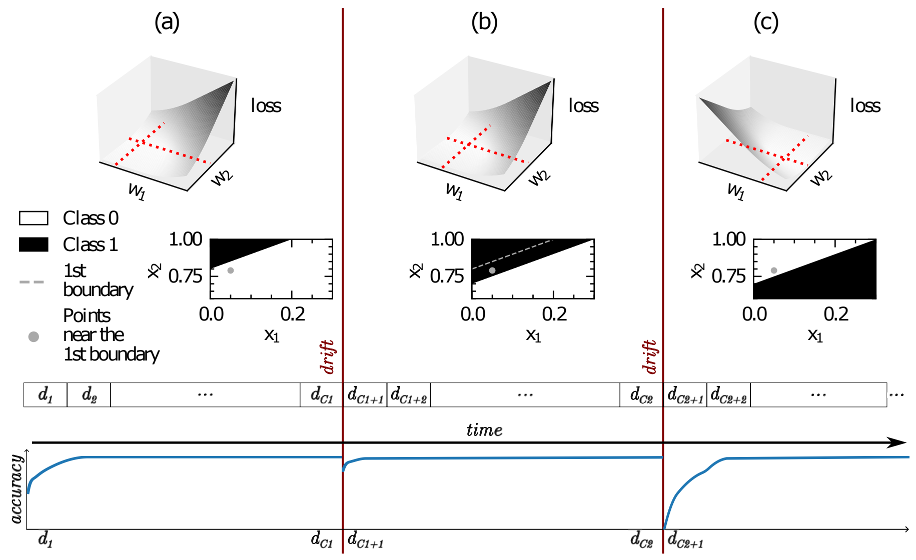

Fig. 2. Loss functions' minimum and accuracy trend of a single linear neuron associated with the following classification functions: (a) $-x_1 + x_2 - 0.8 \geq 0$ (b) $-x_1 + x_2 - 0.7 \geq 0$ (c) $-x_1 + x_2 - 0.7 < 0$.

training set [15]. The data points are, in fact, not available at once, and data has autocorrelations. We, therefore, input the data points in chronological order. Notice that, in this way, we are not minimizing the loss function to all the data but only to the most recently seen data points [1]. Indeed, the literature on data streams [2] commonly assigns greater weight to recent data points because we expect that future data points related to the current concept will bear greater similarity to recent data. In particular, we adopt windowing from Data Stream Management Systems to propose (see Fig. 3) to buffer data points in a batch with size B and build the sequences using a sliding window (with size W) once the batch is complete. In this way, we produce B-W+1 sequences for each batch. Notably, the windowing approach permits us to keep the temporal order.

3.3 Continuous PNN (cPNN)

To better adapt to the concept drift, we propose a methodology to combine the knowledge gained from previous concepts with that learned from the current one. At the same time, we deal with catastrophic forgetting and, thus, provide accurate predictions for all the concepts. Moreover, we handle data points arriving continually from an unbounded data stream and tame temporal dependencies.

PNNs and GIM can recycle old knowledge and avoid CF but are meant to be applied to CL experiences. We, thus, combine SML and CL techniques to build **Continuous PNN (cPNN)**: a continuous version of PNNs. We first define **Continuous LSTM (cLSTM)**, a continuous version of LSTM whose input

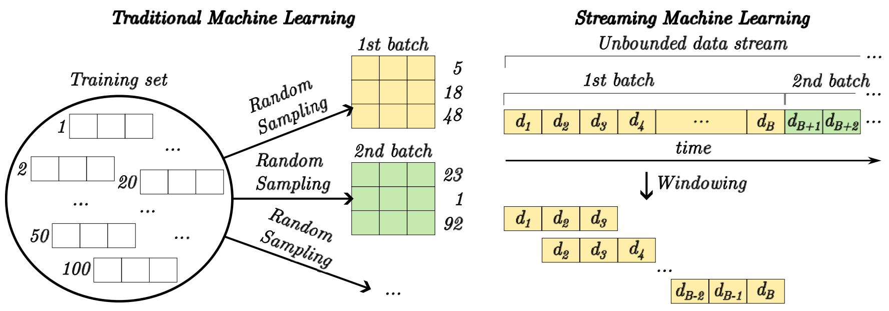

Fig. 3. Data processing in cases of Traditional Machine Learning and SML.

is built as explained in Section 3.2. cLSTM outputs a probability distribution for each sequence item. Each data point's probability distribution on the target classes is computed by averaging its probability distributions associated with all the sequences to which it belongs. We then consider each concept as a task of CL. We use cLSTM as the base model (column) of cPNN to learn continually from an unbounded data stream's data points and tame temporal dependencies. Lateral connections are implemented as suggested by GIM to reduce the parameters. The architecture (shown in Fig. 4) can be edited by changing the column's type from cLSTM to any RNN model.

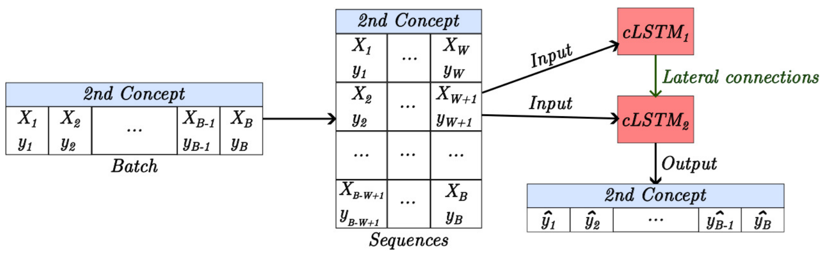

Fig. 4. The cPNN architecture during the second concept training.

Algorithm 1 details the cPNN's lifecycle. The architecture initially has a single column. We buffer the data stream in a batch with size B (Line 5) and create the model's input (Line 9) when the batch is complete. We, then, apply Prequential evaluation [6] (Lines 10-12), taking first the model's predictions and evaluating the performance on the entire batch. Finally, we train the model on the batch for several epochs. After a concept drift, the model receives the batch accumulated up to that time (Line 7). Then, we add a new column to the architecture, building lateral connections and freezing the weights of the previous column (Line 15). Since CL assumes that the label associated with each experience is known and experiences are not mixed, we also assume that drifts are abrupt and to know when they occur. We rely on the presence of a concept label c_t for each data point, which is the same for all the data points in a batch.

Algorithm 1 cPNN training

```
Input: Data stream $S$, Batch size $B$, Epochs $E$, Window Size $W$.
 1: $batch \leftarrow empty\ list$, $perf \leftarrow empty\ list$, $c_{t-1} \leftarrow -1$, $model \leftarrow new\ cPNN()$
 2: for all $(X_t, y_t, c_t)$ in $S$ do
 3:   $drift \leftarrow True$
 4:   if $c_t = c_{t-1}$ then
 5:     Append $(X_t, y_t)$ to $batch$
 6:     $drift \leftarrow False$
 7:   if $|batch| = B$ OR $drift = True$ then
 8:     if $|batch| \geq W$ then
 9:       $X, Y \leftarrow BuildSequences(batch, W)$
10:       $pred \leftarrow model.predict(X)$
11:       Append $Evaluate(pred, Y)$ to $perf$
12:       $model.fit(X, Y, E)$
13:     $batch \leftarrow empty\ list$
14:   if $drift = True$ then
15:     $model.addColumn()$
16:     Append $(X_t, y_t)$ to $batch$
17:   $c_{t-1} \leftarrow c_t$
```

4 Experimental Evaluation

This Section presents our experiments. Section 4.1 explains the generation of the data streams used in the ablation study described in Section 4.2.

4.1 Generated Data Streams

As detailed in [21], the most commonly used SML benchmarks containing temporal dependencies (Electricity and CoverType) are unsuitable for our purpose. The most known synthetic data stream generators (SINE [5], SEA [22], Hyperplane [9], and STAGGER [20]) do not, instead, introduce temporal dependencies in the data. We, thus, propose the construction of a synthetic generator to have a simple and controlled case study to apply the models and analyze their behaviors. We start from SINE and produce a variant whose generated points have temporal dependencies. We begin from a randomly generated two-dimension point in (0,1). Each coordinate of the following points is generated by summing a random value (random walk [17]) to the previously generated point's value. Every random walk's sign is generated to prevent exceeding the range (0,1). After quantifying the autocorrelation between data points using the Partial Autocorrelation Function plot, we set the maximum size of data points having the same label as ten.

To identify the boundaries of the classes, we utilize the two SINE generator's **boundary functions** defined in Equation 3 and 4.

$$S1 : x_1 - sin(x_2) = 0 \quad ⧄ \tag{3}$$

$$S2 : x_1 - 0.5 - 0.3\ sin(3\ \pi\ x_2) = 0 \quad ⊲ \tag{4}$$

We denote by ◤ and ◧ the **classification functions** that classify with "1" the points above, respectively, the S1 and S2 curves, while with "0" the remaining ones. ◢ and ◀ invert the labels of ◤ and ◧ respectively. We generate one data stream for every classification function, each representing one concept and containing 50k data points. Let us introduce the term **sign drift** as the drift where a new concept reverses the labels while maintaining the boundary function (e.g. from ◤ to ◢) or changing it (e.g., from ◤ to ◀). We combine the data streams in two ways. Firstly, **classification inversion drift** produces a single sign drift that keeps the boundary function unchanged (Fig. 2.c). Secondly, **boundary function drift** combines all four concepts' data streams by alternating the boundary functions (Fig. 2.b) and producing one or two sign drifts. By design, more than 50% of the sequences with the same label have a maximum length of five, and labels are balanced. When we change the boundary function without a sign drift (e.g., from ◤ to ◧), 65% of the points keep the same label. If we combine a sign drift with a boundary function drift (e.g., from ◤ to ◀), the percentage drops to 35%. Finally, all the points change their labels after a classification inversion drift (e.g., from ◤ to ◢).

4.2 Experimental Setting

We conduct an ablation study for our hypothesis formulation and compare cPNN with two alternative architectures. **mcLSTM** (Multiple cLSTMs) remove the lateral connections so that each new column does not consider the previous column's hidden layer output. Direct application of the base model **cLSTM** (see Section 3.3) removes, instead, the creation of different columns, resulting in a cPNN with only one column that ignores drifts. Hyperparameter values are chosen as follows after executing the preliminary experiments. Epochs number: 10, window size: 10, batch size: 128, learning rate: 0.1, hidden's layer size: 50.[2] The final performance is computed by averaging the batches. Since the labels are balanced and we do not focus on a particular class, we evaluate the accuracy.

Our hypothesis is that cPNN can adapt to new concepts in an ESTS more quickly than the other two architectures. Additionally, we expect that models can quickly adapt to a new concept if it is similar to the previous one. A sign drift would be more complicated if the model does not learn to invert its past knowledge. We evaluate the final accuracy in four **cases** to verify the two hypotheses for each concept. The first two cases (**[1,50]** and **[1,100]**) analyze how models adapt to the new concept by considering the accuracy at the end of the first 50 and 100 batches after the drift.[3] A reasonably accurate model in the first part of the

[2] Complete source code available at `https://github.com/federicogiannini13/cpnn`

[3] Thresholds represent the number of batches during preliminary experiments, after which the more robust and less robust models achieved good performance.

Table 1. Accuracies on classification inversion drift. cPNN outperforms the ablated versions in all cases.

		◤ ◢			◀ ◀		
concept	**case**	**cPNN**	**cLSTM**	**mcLSTM**	**cPNN**	**cLSTM**	**mcLSTM**
2nd	[1,50]	**.96, .004**	*.872, .026*	.903, .008	**.864, .017**	.742, .024	.75, .006
	[1,100]	**.972, .002**	*.921, .014*	.933, .004	**.888, .02**	.79, .024	*.774, .006*
	(100,)	**.989, .001**	.98, .002	*.975, .001*	**.927, .018**	.883, .029	*.834, .009*
	[1,)	**.984, .001**	.965, .004	.964, .001	**.917, .018**	.859, .026	*.818, .007*

concept is robust to concept drifts. The third case (**[100,)**), which covers the batch range from 100 onwards, assesses the accuracy of models in response to the newly introduced concept once they have adapted to it. Finally, the fourth case (**[1,)**) monitors the entire concept by investigating accuracy from the first batch onwards. Each experiment is repeated ten times, and their average accuracy is analyzed. Tables 1, 2 and 3 of Section 5 report results using the mentioned notations.

5 Results

This section analyzes the results of the different experiments described in Section 4.2. Tables 1, 2 and 3 report the ten executions' average accuracies and standard deviations. Since the first concept's architectures are the same, we make comparisons from the second concept onwards. Architectures are compared in pairs. We report in bold the statistically best-performing architecture (if it is statistically better performing than the remaining two) and in italics the less-performing one. We, thus, first conduct a Shapiro-Wilk test to check for normality. If we cannot reject the null hypothesis for both distributions, we conduct a Welch's t-test. Otherwise, we run a Wilcoxon signed-rank test. We perform a one-sided test in both cases. We underline the not normally distributed samples. All the tests are conducted with a significance level of 0.05.

5.1 Classification Inversion Drift

Results in Table 1 show that after the concept drift, cLSTM performance collapse. Since they are similar, we only report results for two data streams. For the S1 classification function, the mcLSTM's random initialization of the parameters works better than the cLSTM one (which is the inverse concept's optimal one), but from the 100th till the end of the concept, cLSTM outperforms mcLSTM. cPNN can adapt quickly to the new concept. It results in being the best-performing model in all the experiments. In the case of S2, the gap between cPNN and the other models is more significant. These experiments suggest that cPNN could learn to invert past knowledge. cLSTM requires more iterations to reach the new optimal setting since it starts from the inverse concept one. At the end of each concept, cLSTM's new optimal configuration is still worse than cPNN's one.

Table 2. Accuracies on data streams ◤, ◧, ◢, ◨ and ◤, ◨, ◢, ◧. cPNN always recovers faster from concept drifts than the ablated versions. In some cases, a single cLSTM performs better in the long run, but in the end, it only remembers that last concept since it does not manage CF. mcLSTM that does not use transfer learning and resets the parameter configuration performs worse in almost all situations.

		◤ ◧ ◢ ◨			◤ ◨ ◢ ◧		
concept	**case**	**cPNN**	**cLSTM**	**mcLSTM**	**cPNN**	**cLSTM**	**mcLSTM**
2nd	[1,50]	**.759, .012**	.746, .011	*.738, .006*	**.781, .006**	*.743, .015*	.753, .006
	[1,100]	**.803, .007**	.791, .012	*.762, .005*	**.808, .004**	.772, .008	.775, .004
	(100,)	.877, .006	**.897, .013**	*.829, .011*	**.876, .005**	.851, .011	*.823, .009*
	[1,)	.858, .005	**.87, .012**	*.812, .009*	**.859, .003**	.831, .009	*.811, .007*
3rd	[1,50]	**.951, .004**	.893, .022	.905, .008	**.947, .004**	.94, .009	*.908, .004*
	[1,100]	**.964, .004**	.932, .013	.935, .004	**.961, .003**	.955, .006	*.936, .003*
	(100,)	.982, .002	.982, .001	*.975, .001*	.981, .002	.982, .001	*.975, .001*
	[1,)	**.977, .002**	.969, .004	*.965, .001*	.976, .002	.975, .002	*.965, .001*
4th	[1,50]	**.855, .012**	.778, .016	*.754, .005*	**.862, .008**	.777, .025	*.738, .005*
	[1,100]	**.88, .011**	.822, .012	*.776, .004*	**.88, .006**	.831, .019	*.76, .003*
	(100,)	.907, .007	.91, .009	*.826, .01*	.909, .009	.915, .013	*.827, .009*
	[1,)	**.9, .008**	.888, .009	*.813, .008*	.902, .008	.894, .014	*.81, .007*

Table 3. Accuracies on data streams ◧, ◤, ◨, ◢ and ◧, ◢, ◨, ◤.

		◧ ◤ ◨ ◢			◧ ◢ ◨ ◤		
concept	**case**	**cPNN**	**cLSTM**	**mcLSTM**	**cPNN**	**cLSTM**	**mcLSTM**
2nd	[1,50]	.931, .014	.933, .012	*.915, .004*	**.921, .008**	.892, .022	.903, .003
	[1,100]	.943, .01	**.951, .008**	.941, .003	.939, .009	.934, .012	.934, .003
	(100,)	*.974, .003*	**.983, .001**	.976, .001	*.971, .004*	**.983, .001**	.975, .001
	[1,)	.966, .005	**.974, .003**	.967, .001	.963, .005	**.97, .003**	.964, .001
3rd	[1,50]	**.824, .013**	.768, .023	.754, .005	**.835, .013**	.779, .016	*.755, .007*
	[1,100]	**.851, .015**	.802, .024	*.774, .004*	**.863, .01**	.812, .018	*.776, .003*
	(100,)	.896, .01	.892, .018	*.835, .011*	.903, .009	.893, .017	*.831, .01*
	[1,)	**.885, .011**	.869, .02	*.819, .009*	**.893, .009**	.872, .017	*.817, .008*
4th	[1,50]	**.952, .006**	.942, .007	*.907, .008*	**.953, .007**	.923, .017	.92, .004
	[1,100]	**.963, .004**	.957, .003	*.936, .005*	**.962, .004**	.945, .01	.944, .002
	(100,)	.98, .004	**.983, .002**	*.975, .001*	.979, .003	**.982, .002**	*.976, .001*
	[1,)	.975, .004	.976, .001	*.965, .002*	.975, .003	.973, .004	*.967, .0*

5.2 Boundary Function Drift

Results regarding boundary function drift (shown in Tables 2 and 3) indicate that cPNN adapts more quickly to a new concept after a sign drift and when the new boundary function is more complex than the previous (a drift from S1 to S2). In this case, cPNN outperforms the other architectures in the first 50 and 100 batches. From the 100th batch, cLSTM and cPNN have similar performance. cLSTM outperforms cPNN in the first batches only after the first drift from S2 to S1, with no sign drift. mcLSTM performs worse in almost all the experiments.

6 Conclusion

This paper pioneers a novel continuous version of PNNs for Evolving Streaming Time Series. We proposed CPNN to deal simultaneously with concept drifts and temporal dependencies while avoiding catastrophic forgetting. To do so, we presented a continuous adaptation of LSTM (namely cLSTM) that exploits the SGD algorithm to tame temporal dependencies in a data stream. A similar method was used by [11] on a complex architecture and real datasets. Instead, our goal was to analyze the models' behaviors using a simplified scenario. cPNN's architecture is based on PNNs to tame CF and use transfer learning to fit new concepts quickly. To investigate cPNN behavior, we generated synthetic data streams and conducted an ablation study. cPNN performance highlighted a quicker adaptation to new concepts. Its average accuracy after each concept drift is, in fact, statistically greater than the ablated ones. cPNN resulted, thus, in being more robust to concept drifts, especially in the case of sign drift.

One of the main limitations of cPNN is that its complexity increases linearly with the number of concepts. We, thus, imagine that this architecture could be applied in the case of reoccurrent drifts where we would need to check whether the new concept has been seen before. Additionally, when dealing with data streams, the selection of hyperparameters can become challenging, and the resulting outcomes may be highly sensitive to these choices. Moreover, we only studied the models in a simplified scenario with abrupt concept drifts and synthetic data streams containing only two features. In our future works, we intend to explore more types of drift in a higher dimensional space and complex classification functions. Finally, as in many CL experiments, we assumed to have an "oracle" that knows the concept associated with each data point. In our future works, we will apply concept drift detection methods. cPNN performance results suggested that it could automatically learn to invert past knowledge when there is a sign drift. We also think its quicker adaptation to the new concept is due to past recycling ability. We will analyze the model's parameters in future works to verify it. In the long term, we intend to investigate how cPNN learns in contexts where real data evolves via gradual or incremental concept drifts. We will most likely need to examine other types of columns, like Gated Recurrent Units or Transformers.

References

1. Anagnostopoulos, C., Tasoulis, D.K., Adams, N.M., Pavlidis, N.G., Hand, D.J.: Online linear and quadratic discriminant analysis with adaptive forgetting for streaming classification. Stat. Anal. Data Min. **5**(2), 139–166 (2012)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: PODS. pp. 1–16. ACM (2002)
3. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine learning for data streams: with practical examples in MOA. MIT press (2018)
4. Cossu, A., Carta, A., Bacciu, D.: Continual learning with gated incremental memories for sequential data processing. In: IJCNN. pp. 1–8. IEEE (2020)

5. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: SBIA. LNCS, vol. 3171, pp. 286–295. Springer (2004)
6. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: KDD. pp. 329–338. ACM (2009)
7. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
9. Hulten, G., Spencer, L., Domingos, P.M.: Mining time-changing data streams. In: KDD. pp. 97–106. ACM (2001)
10. Lange, M.D., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G.G., Tuytelaars, T.: A continual learning survey: Defying forgetting in classification tasks. IEEE Trans. Pattern Anal. Mach. Intell. **44**(7), 3366–3385 (2022)
11. Lemos Neto, Á.C., Coelho, R.A., Castro, C.L.d.: An incremental learning approach using long short-term memory neural networks. Journal of Control, Automation and Electrical Systems pp. 1–9 (2022)
12. Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., Rodríguez, N.D.: Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. Inf. Fusion **58**, 52–68 (2020)
13. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: A review. IEEE Trans. Knowl. Data Eng. **31**(12), 2346–2363 (2019)
14. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, vol. 24, pp. 109–165. Elsevier (1989)
15. Meng, Q., Chen, W., Wang, Y., Ma, Z., Liu, T.: Convergence analysis of distributed stochastic gradient descent with shuffling. Neurocomputing **337**, 46–57 (2019)
16. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
17. Pearson, K.: The problem of the random walk. Nature **72**(1865), 294–294 (1905)
18. Read, J., Rios, R.A., Nogueira, T., de Mello, R.F.: Data streams are time series: Challenging assumptions. In: BRACIS (2). Lecture Notes in Computer Science, vol. 12320, pp. 529–543. Springer (2020)
19. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. CoRR **abs/1606.04671** (2016)
20. Schlimmer, J.C., Granger, R.H.: Incremental learning from noisy data. Mach. Learn. **1**(3), 317–354 (1986)
21. de Souza, V.M.A., dos Reis, D.M., Maletzke, A.G., Batista, G.E.A.P.A.: Challenges in benchmarking stream learning algorithms with real-world data. Data Min. Knowl. Discov. **34**(6), 1805–1858 (2020)
22. Street, W.N., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: KDD. pp. 377–382. ACM (2001)
23. Ziffer, G., Bernardo, A., Della Valle, E., Cerqueira, V., Bifet, A.: Towards time-evolving analytics: Online learning for time-dependent evolving data streams. Data Science (Preprint), 1–16
24. Zliobaite, I., Bifet, A., Read, J., Pfahringer, B., Holmes, G.: Evaluation methods and decision theory for classification of streaming data with temporal dependence. Mach. Learn. **98**(3), 455–482 (2015)