

Addressing Temporal Dependence, Concept Drifts, and Forgetting in Data Streams

Federico Giannini¹[0000–0002–4210–6271] and Emanuele Della
Valle¹[0000–0002–5176–5885]

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

Abstract. Several challenges arise when applying Machine Learning in data streams. Firstly, data points are continuously generated, and algorithms should continuously learn from each. The solution should also promptly adapt to changes in data distribution (known as concept drifts). Additionally, while addressing concept drifts, preserving the knowledge gained from past data is crucial to avoid the problem of catastrophic forgetting. Finally, one should consider the temporal dependence that data points may exhibit. Three communities address these problems separately: Streaming Machine Learning (SML), Continual Learning (CL), and Time Series Analysis (TSA). In our previous research, we proposed Continuous Progressive Neural Networks (cPNN), a first approach that considers all the challenges together. It bridges SML, CL, and TSA by producing a continuous adaptation of the CL strategy of Progressive Neural Networks. It uses transfer learning to adapt to changes quickly and tames temporal dependence using Long Short-Term Memory. In this work, we present a comprehensive experimental campaign that analyzes the behaviour of SML models and cPNN in the case of complex temporal dependence and various concept drifts on synthetic and real data streams. Results bring statistical evidence that SML models struggle with substantial temporal dependence, while cPNN is a viable solution.

Keywords: Data streams · Concept drifts · Temporal dependence.

1 Introduction

Applying Machine Learning models to classify data points brings about a unique set of challenges in a scenario where data is generated from an unbounded data stream [3]. Firstly, a one-time learning phase on the entire dataset is impossible as the data is not available at once. New data points arrive continuously, and the model must learn from them. We want, therefore, to continuously train the model on single mini-batches or even a single data point at a time. Secondly, the distribution of data may change over time. This phenomenon is known as *concept drift* [17]. The goal is quickly adapting to changes to avoid performance slumps after a concept drift. Furthermore, data points may be autocorrelated, with each data point potentially relying on the preceding ones at a given timestamp [25].

Considering this dependence, it's crucial to solve the classification tasks efficiently. Finally, when a concept drift occurs, the model must learn the new distribution, which can lead to losing its predictive ability on previous ones. The stability-plasticity dilemma [19] regulates this phenomenon where plasticity is the ability to learn new knowledge, while stability is remembering the past acquired one. Too much plasticity could lead to *catastrophic forgetting* [14]. Conversely, too much stability can lead to difficulty in learning new knowledge. This problem may be particularly relevant when old distributions reoccur over time, or past knowledge may be useful to solve the problem associated with the current distribution.

These challenges are addressed by three main communities. Streaming Machine Learning (SML) [3] aims to enable continuous learning and adapt quickly to concept drifts. However, it assumes that data is independent and identically distributed (shortly i.i.d.) between two concept drifts. It, thus, fails to account for temporal dependence. Additionally, forgetting is usually ignored. Although Time Series Analysis [5], and particularly Recurrent Neural Networks (RNN), offer a potential solution for the former, they are poorly used in the streaming context. Moreover, neural networks may catastrophically forget the knowledge of previous concepts. Long Short-Term Memory (LSTM) [13] is one of this domain's most commonly used recurrent RNN architectures. Finally, Continual Learning (CL) [16] focuses on avoiding forgetting in Deep Learning models but assumes the data stream to be composed of large batches of data points accessible simultaneously. In this context, Progressive Neural Networks (PNN) [23] is a strategy to deal with forgetting and uses transfer learning to recycle old knowledge. Given these premises, Continuous Progressive Neural Networks (cPNN) [9] acts like a bridge between SML, CL and TSA and represents the first pioneering solution to solve all these challenges simultaneously. It tames temporal dependence using cLSTM, a continuous version of LSTM that buffers the data stream in fixed-size mini-batches and builds sequences on each. It then applies a PNN on top of cLSTM, allowing the solution to avoid forgetting and exploit transfer learning to adapt to new concepts quickly. cPNN is a *periodic classifier* that requires mini-batches of data points to perform inference and training.

This work presents a complete analysis of SML models and cPNN to answer the following **research questions**: *How do SML models and cPNN react to different types of drifts? Are they able to tame temporal dependence?* We set up an experimental campaign involving synthetic and real data streams. Additionally, the classification problems implicate elaborated temporal dependence on features and target labels. We compare cPNN and its ablated versions with classical SML models. We also propose a simple way to perform this comparison in a scenario requiring classification to be performed whenever a new data point is generated. In this reproducibility paper, we, therefore, repeat the experiments of our original cPNN paper [9] using the original source code in new contexts (we add new datasets and new models to the benchmark) to generalize our previous findings.¹

¹ The code and the data used in this work are available here for reproducibility: https://github.com/federicogiannini13/cPNN_extended

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

2 Related Works

In this work, we consider three main research areas focused on the challenges of data streams. Fig. 1 summarizes the models and their associated areas.

In a data stream classification problem, a data stream is an unbounded sequence of data points $D : d_0, d_1, \dots, d_t, d_{t+1}, \dots$ with $t \in \mathbb{N}$. Each data point d_t is a tuple $\langle X_t, y_t \rangle$ where X_t is the feature vector and y_t is the associated label. The assumption is that, after receiving the feature vector X_t , the model must predict the associated label \hat{y}_t . The correct label y_t will be available after the prediction and before receiving the new feature vector X_{t+1} . This way, one can apply a *prequential evaluation* mechanism [8] that, whenever a new data point d_t is generated, acts as follows: 1) Infer the label \hat{y}_t by inputting X_t to the model. 2) Update a performance metric using the correct label y_t . 3) Update the model using $\langle X_t, y_t \rangle$.

In contrast to classical Machine Learning, which assumes that data is independent and identically distributed, a crucial issue is *concept drift* [17]. Data can, in fact, change its distribution over time. A *concept* is the unobservable random process that produces the data points [7]. The concept drift is *virtual* when there is a change in the probabilities $P(X|y)$ or $P(y)$. Notably, this type of change does not affect the class boundary. Conversely, *real concept drifts* affect the probability $P(y|X)$, changing the class boundary. Additionally, concept drifts can be categorized by considering the speed at which they occur. When an *abrupt* concept drift occurs at time t , the concept immediately changes from $t - 1$ to t . The new concept gradually or incrementally replaces the previous during *gradual* or *incremental* concept drifts. Concepts can also re-occur over time. Finally, data can exhibit *temporal dependence* when dealing with data streams. In this circumstance, given a data point d_t , $\exists \tau P(a_t|b_{t-\tau}) \neq P(a_t)$ where $a_t \in X_t \cup \{y_t\}$, $b_{t-\tau} \in X_{t-\tau} \cup \{y_{t-\tau}\}$.

SML [3] develops concept drift detectors to detect concept drift automatically [17] and proposes algorithms to adapt to these changes quickly. Tree-based models implement streaming versions of decision trees that incrementally select the node. *Hoeffding Adaptive Tree (HAT)* [2] incorporates the ADaptive WINdowing (ADWIN) change detector [1] for concept drift. Ensemble methods combine predictions using voting strategies. *Adaptive Random Forests (ARF)* [11] use the Poisson distribution for resampling and training on diverse samples. SML usually assumes data points within a concept to be temporal independent. However, *Temporal Augmentation (TA)* [4] has emerged as the first meta-strategy to take temporal dependence into account. Denoting the order by o , TA adds to the feature space of each data point the labels of the previous o data points.

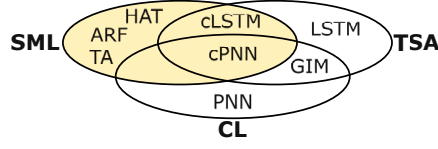


Fig. 1: Summarization of the models considered in this work and their research areas.

Conversely, CL addresses the forgetting problem when learning from data streams using deep-learning models. It usually assumes the data stream to be split into large batches of data points called experiences. In this scenario, data points do not arrive individually, but a new experience arrives at each timestamp. Each experience introduces a new concept. The goal is to incorporate the knowledge from the new experience without forgetting the previous one. PNNs [23] is a famous solution in this context. During the training on the first experience, a PNN contains a single neural network (called *column*). When a new experience arises, PNN reuses the knowledge gained during previous experiences that could be useful in solving the current problem. To do so, it exploits transfer learning and connects the hidden layers of the different columns. The output of hidden layer i of the column k is computed as in Equation 1, where W_i and U_i are the weight matrices to be learned. U_i implement transfer learning. When adding a new column, PNN freezes the weights of the previous ones to avoid forgetting. Since, during the inference, it is supposed to know which experience is associated with a specific data point, PNN can select the corresponding column output.

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

Gated Incremental Memories (GIM) [6] introduces a recurrent version of PNN to tame temporal dependence. It uses LSTM [13], a famous deep-learning model in TSA that works with fixed-size sequences of items. Recent advancements highlighted the forecasting potential of Deep Learning [18]. GIM simplifies PNN by connecting each column only to the preceding one. As expressed by Equation 2, given a sequence of items and a column k , the new feature vector $X_j^{(k)}$ for the j -th item of the sequence concatenates the original feature vector X_j with the LSTM hidden layer's output for the item in column $k - 1$.

$$X_j^{(k)} = X_j \parallel LSTM_j^{(k-1)} \quad (2)$$

GIM works with CL experiences where each sample is a sequence of items, and the different samples within an experience are independent. To address the challenges of continuously learning from mini-batches of data points, managing temporal dependence, handling concept drifts, and preventing forgetting we introduced **Continuous Progressive Neural Networks (cPNN)** [9]. As Fig. 2 shows, it uses cLSTM (a continuous extension of LSTM originally presented

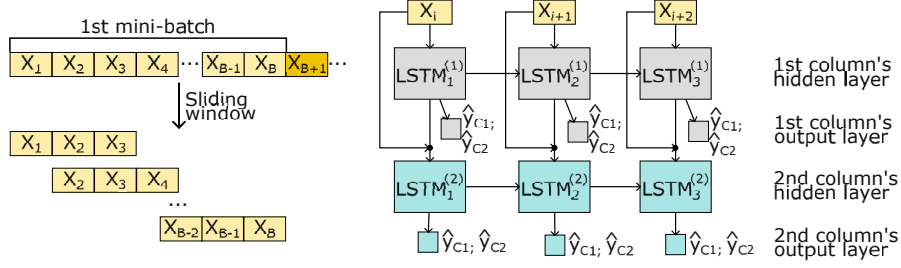


Fig. 2: For each sequence item, cPNN computes scores for each class, averaging scores for the same data point across sequences. Each column concatenates, for each item j , the feature vector X_j and the associated previous column's hidden layer's output.

by Neto et al. [15] as GIM's columns. cLSTM tames temporal dependence when learning continuously from a data stream by accumulating data points in fixed-size mini-batches. When the mini-batch is complete, it builds sequences using a sliding window. Then, it trains a many-to-many LSTM model on the sequences. Given each sequence item, cLSTM outputs a score for each target class. The score $\hat{y}_c(d_t)$ for a given target class c on data point d_t is determined by averaging the scores $\hat{y}_c^s(d_t)$ from all sequences s containing d_t within the mini-batch. For each mini-batch, the loss function is computed as the Binary Cross Entropy, where the prediction for a given target class c on data point d_t is obtained as expressed in Equation 3.

$$\hat{y}_c(d_t) = \text{Mean}\{\hat{y}_c^s(d_t) | d_t \in s\} \quad (3)$$

cLSTM and cPNN are, thus, periodic classifiers that require mini-batches of data points during both the training and inference phases. This approach can be a reasonable option in various practical situations. As expressed in [22], there are many cases where it is unnecessary to produce predictions immediately. Additionally, online true single-pass learning is rarely preferable and sometimes critical. When the model has seen enough data, the knowledge from new data points can be incorporated into the model with less urgency. Moreover, in a data stream scenario, there are only a few rare cases in which a single training overwhelms the memory of a device. In most cases, storing data points in a buffer is possible by limiting the size of the buffer. Furthermore, utilizing mini-batches can enhance the model's representational capacity when possible. The label for each data point is determined by examining its complete history within the mini-batch, from when the data point is the sequence's final item to when it is the initial one. This allows us to consider both past and future temporal dependence.

Applying GIM prevents forgetting by freezing the weights of the columns associated with the previous concepts. It also exploits transfer learning to boost adaptation after a concept drift by reusing past knowledge. The model combines the knowledge associated with the previous column with the feature coming

from the data points. During the training procedure, it learns how to weigh the two. Algorithm 1 outlines the lifecycle of the cPNN [9]. Initially, the architecture consists of a cLSTM (column). The incoming data stream S is buffered into mini-batches of size B (Line 7). Once the mini-batch is complete, a sliding window builds the sequences of size W (Line 10). Next, prequential evaluation [8] is used (Lines 11-13), where the model’s predictions are first obtained, and then its performance is assessed on the entire mini-batch. Subsequently, the model is trained on the mini-batch over E epochs. Upon detecting a concept drift, the model processes the accumulated batch (Line 8). After freezing the weights of the last column C , cPNN introduces a new column and connects it to C (Line 16).

Algorithm 1 cPNN training

Input: Data stream S , Batch size B , Epochs E , Window Size W .

```

1: batch  $\leftarrow []$ 
2: perf  $\leftarrow []$ 
3: model  $\leftarrow$  new cPNN()
4: drift  $\leftarrow$  False
5: for all  $(X_t, y_t)$  in  $S$  do
6:   if drift = False then
7:     batch.append $((X_t, y_t))$ 
8:   if len(batch)= $B$  OR drift=True then
9:     if len(batch)  $\geq W$  then
10:       $X, Y \leftarrow$  BuildSequences(batch,  $W$ )
11:      pred  $\leftarrow$  model.predict( $X$ )
12:      perf.append(Evaluate(pred,  $Y$ ))
13:      model.fit( $X$ ,  $Y$ ,  $E$ )
14:      batch  $\leftarrow []$ 
15:   if drift=True then
16:     model.addColumn()
17:     batch.append $((X_t, y_t))$ 
18:   drift  $\leftarrow$  detectDrift( $X_t, y_t$ )

```

3 Experimental Setting

As highlighted by our cPNN original paper [9], if an abrupt concept drift introduces a classification problem similar to the previous, the Stochastic Gradient Descent (SGD) may be able to adapt to the new concept quickly. Conversely, when the new concept substantially changes the classification problem, SGD may require more steps to converge since it starts from the optimal configuration of the previous concept that could be far from the new one. The goal of cPNN is to boost the adaptation to the new concept by exploiting transfer learning and reusing useful past knowledge. The first objective of this work is to analyze the behaviour of cPNN in the cases of different types of drifts and compare it with

SGD on data containing elaborated temporal dependence. Additionally, we are interested in analyzing the behaviour of SML models in the case of temporal dependence. We also introduce the usage of TA to check whether it improves the performance of SML models. Finally, we compare cPNN with SML models.

This Section describes the experimental setting, with Section 3.1 explaining the data streams generation, Section 3.2 presenting the considered models, and Section 3.3 illustrating the hypotheses formulation.

3.1 Data Streams Generation

The most known benchmarks with temporal dependence, as highlighted in [9,24], are unsuitable for our needs. Moreover, existing synthetic data stream generators lack temporal dependence. Hence, we suggest two new generators for synthetic and realistic data. We use the term *boundary function* to indicate a function that determines the boundaries between the classes of a specific classification problem. A *classification function* is, instead, a function that assigns the labels to the points given a boundary function. We indicate as *severe drift* a concept drift where a new concept reverses the labels while maintaining or changing the boundary function. Otherwise, we refer to the drift as *mild drift*. We consider two boundary functions and, thus, four classification functions. Each classification function represents a concept associated with thousands of data points. We combine the four concepts in four different *configurations* by alternating the boundary functions and producing three abrupt concept drifts.

We propose two new generators. *SRWM* generates synthetic data points and produces elaborated temporal dependence on both features and labels. *Weather* approaches a realistic scenario by using data collected from weather sensors.

SRWM generator The first generator extends the SineRW generator (*SRW*) presented in [9], which generates points in two dimensions (x_1 and x_2) via a random walk process. It starts with a randomly generated two-dimensional point within the range (0,1). To generate the feature values of a given point, a random walk is performed, where a random value is incrementally added to the previous value [21]. These random walks are drawn from a specific distribution and have a maximum magnitude of 0.05, with a randomly assigned sign. The sign is reversed to prevent feature values from exceeding the (0,1) range if adding the random walk could cause the new value to go beyond this range. In this manner, each feature can be viewed as a time series. To identify the boundaries of the classes, it utilizes the two SINE generator’s boundary functions [7] defined in Equation 4. Each boundary function results in two binary classification functions: one assigns 1 to points meeting the condition obtained by replacing the equality with the inequality ≥ 0 , while the others are assigned 0. The second uses < 0 . \square generates \blacksquare and \blacktriangle , while \boxless generates \blacksquare and \blacktriangleleft . Examples of severe drifts are from \blacksquare to \blacktriangle or from \blacktriangle to \blacksquare . Examples of mild drifts are from \blacksquare to \blacktriangle or from \blacktriangle to \blacksquare .

$$\square : x_1 - \sin(x_2) = 0 \quad \boxless : x_1 - 0.5 - 0.3 \sin(3 \pi x_2) = 0 \quad (4)$$

SRW generator injects temporal dependence in the features of the data points. However, the label of a specific data point can be inferred by looking only at that single data point, without considering sequences. To complicate the classification problem, we incorporate temporal dependence into the data points' labels and produce SineRW Mode (SRWM). The updated label at time t , represented as y'_t , is computed using Equation 5, where y_t denotes the binary label assigned by SRW. We strengthen the temporal connections between data points by considering also the past labels. This way, a data point's label cannot be inferred only using its features. For each concept, we generate 50k data points.

$$y'_t = MODE(y_t, y_{t-1}, y_{t-2}, y_{t-3}, y_{t-4}) \quad (5)$$

Weather Generator The second generator starts from the Weather dataset [10] to consider realistic scenarios. Weather is a dataset from the Agricultural Research Service, the U.S. Department of Agriculture's chief scientific in-house research agency. It contains detailed hydrometeorological data from the mountain rain-to-snow transition zone from 2004 through 2014 from the Johnston Draw watershed ($1.8km^2$) in southwestern Idaho. Data includes continuous hourly hydrometeorological variables from two stations across a $372m$ elevation gradient on the north- and south-facing slopes, resulting in 96432 data points for each station. More precisely, we use the following features: A : Air temperature in $^{\circ}C$, RH : Relative Humidity in percentage, w_s : Wind speed in ms^{-1} , w_d : Wind direction from 0° to 360° , T_d : Dew point Temperature in $^{\circ}C$. Water vapor pressure is not considered due to its perfect positive correlation with the dew point temperature T_d . We replicate a classification problem by using a specific feature to construct binary labels. We choose this feature after conducting an analysis of the correlations between the features and selecting the most correlated one. In this way, we produce a classification problem that allows for the prediction of the label using the remaining features. We select the first station and choose A as the target feature since it has the following Pearson's r correlation coefficients: 0.7 with RH , 0.6 with T_d , -0.2 with w_s , and -0.3 with w_d . We scale the different features to facilitate the deep-learning learning process.

After choosing the target feature, we create two binary classification functions as expressed in Equations 6 and 7 to incorporate elaborate temporal dependence. X_t is the feature vector of the data point at timestamp t , A_t is the value of the target feature. We obtain two further functions (F_{1-} , F_{2-}) by inverting the labels of F_{1+} , and F_{2+} . We can imagine two boundary functions: F_1 compares the current data point with the previous one, F_2 with the median of the previous ten. Examples of severe drifts are from F_{2+} to F_{1-} or from F_{1+} to F_{2-} . Examples of mild drifts are from F_{1+} to F_{2+} or from F_{2-} to F_{1-} . The data set is split into four concepts of the same length, each assigned to a classification function.

$$F_{1+} : y(X_t) = \begin{cases} 1, & \text{if } A_t > A_{t-1} \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

$$F_{2+} : y(X_t) = \begin{cases} 1, & \text{if } A_t > \text{Median}(A_{t-10}, \dots, A_{t-1}) \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

3.2 Models and Scenarios

Our experiments compare cPNN and SML models with and without TA. We label SML models with TA as SML_T , and the classical ones without TA as SML_C . We use two SML_C models: ARF and HAT . Additionally, we create variants with TA: ARF_T and HAT_T . We also test the following *ablated versions* of cPNN to investigate the need for its different components (as also done by [9]):

- *cLSTM*: it directly applies cLSTM without the GIM architecture. After a concept drift, it continues training on the new concept, starting from the optimal parameters of the previous concept. Therefore, it does not avoid forgetting and is unaware of concept drift.
- Multiple cLSTMs (*mcLSTM*): a cPNN in which each column does not consider the previous column’s hidden layer output. It avoids, thus, forgetting but does not use transfer learning.

We must consider that SML and cPNN models work differently. cPNN and its ablated versions perform inference and training after accumulating a mini-batch of data points. We call this setting **periodic scenario**. Conversely, we call **anytime scenario** the setting where classification must be performed whenever a new data point is generated. While SML models can perform learning on each data point, cLSTM (and thus mcLSTM and cPNN) learning is designed for mini-batch processing and cannot be altered. Traditional deep-learning models typically run training on fixed-size mini-batches rather than individual data points. This approach facilitates the Stochastic Gradient Descent algorithm to produce more precise gradient estimates and enables parallel processing of data points [12]. We preserve the training phase with mini-batches intact. Nevertheless, we can adjust the inference phase to operate as an anytime classifier. Instead of waiting for the mini-batch to be complete before predicting each data point, cLSTM can instantly predict each data point as generated in the data stream. To achieve this, cLSTM can rely solely on the initial sequence in which the data point appears. Let W be the window size. To classify the data point at time t , the model must buffer the data points from $t - W + 1$ to $t - 1$. This way, cLSTM keeps a **periodic learner** but becomes an **anytime classifier** like SML models and can, thus, make predictions whenever a new data point is generated.

cPNN models hyperparameters are chosen as follows after executing the preliminary experiments (as in our original cPNN paper [9]). Epochs number **E**: 10, mini-batch size **B**: 128, learning rate: 0.1, hidden layer size: 50. For Weather, the window size is set to 11 to incorporate the temporal dependence of F_2 . SML models use the default parameters, while the order of TA is set to consider the same number of data points of cPNN models. To focus on the models’ reaction to drifts, we are supposed to know the exact time of concept drifts. This can result from applying a concept drift detector with 100% accuracy.

3.3 Hypotheses formulation













We formulate the following research hypotheses:

- **H1:** cPNN adapts to new concepts more quickly than its ablated versions if the drift is severe. Verification: cPNN statistically outperforms its ablated versions after encountering the first part of a concept following a severe drift.
- **H2:** cPNN adapts to new concepts more quickly than its ablated versions if the drift introduces an already-seen boundary function. Verification: cPNN statistically outperforms its ablated versions in the initial part of a new concept that introduces an already-seen boundary function.
- **H3:** cLSTM quickly adapts to a new concept similar to the previous one. Verification: the cPNN performance is not significantly better than cLSTM after encountering a new boundary function following a mild drift.
- **H4:** an SML_C model cannot learn in temporal dependence. Verification: after the initial part of the concepts and upon its end, each SML_C model is statistically outperformed by at least one SML_T and one cPNN model.
- **H5:** SML_T models improve the taming of temporal dependence. Verification: each SML_T model statistically outperforms its version without the TA after the initial part and the end of the concepts.
- **H6:** cPNN models outperform SML_T models, even though the learning phase could be slower. Verification: cPNN or one of its ablated version statistically outperform all other models at the end of each concept.

The comparison between cPNN models is performed in the periodic scenario since it's where cPNN is designed to operate. The training goal is, in fact, not aligned with the inference one in the anytime scenario since the training loss function is computed by averaging the scores of the same data point over a mini-batch. In contrast, inference is computed on single data points. We, thus, verify H1, H2, and H3 in a periodic scenario. H4, H5, and H6 are verified in an anytime scenario since SML models are meant to deal with it.

Since labels can be unbalanced, we use Cohen's kappa score as a performance metric. Cohen's Kappa is a statistical measure used to quantify agreement between two raters [20]. It is usually applied to evaluate the agreement between the true labels and those a learned model assigns. It is calculated as $K = \frac{P_0 \cdot P_e}{1 - P_e}$ where P_0 is the observed agreement, and P_e is the expected agreement by chance. Values range from -1 to 1, with 1 indicating perfect agreement, 0 indicating no agreement beyond chance, and negative values indicating disagreement. The interpretation of Cohen's Kappa values is generally as follows: (0, 0.2) slight agreement, [0.2, 0.4) fair agreement, [0.4, 0.6) moderate agreement, [0.6, 0.8) substantial agreement, [0.8, 1) almost perfect agreement. We compute it using the prequential evaluation [8]. For each model in the anytime scenario, whenever a new data point d_t is generated, we take the prediction, update the score and then update the model. Scores are concept-specific and reset after concept drifts. The score at time t considers all the predictions made by the model from the first data point following the last drift to d_t . In contrast, whenever a mini-batch b_t is filled up in the periodic scenario, we take the predicted labels of all the

Table 1: Average Cohen’s Kappa scores of the cPNN ablation study in the periodic scenario. M stands for a concept that introduces mild drift, and S for severe drift.

	SRWM						Weather					
	(from the 2nd concept onwards)						(from the 2nd concept onwards)					
	start	end	start	end	start	end	start	end	start	end	start	end
	 (M)		 (S)		 (M)		F_{2+} (M)		F_{1-} (S)		F_{2-} (M)	
cLSTM	.43	.60	.72	.86	.44	.61	.70	.76	.30	.54	.74	.79
cPNN	.40	.61	.83	.90	.62	.72	.67	.76	.69	.73	.81	.83
mcLSTM	.33	.49	.67	.84	.34	.49	.39	.59	.23	.52	.43	.60
	 (S)		 (M)		 (S)		F_{2-} (S)		F_{1-} (M)		F_{2+} (S)	
cLSTM	.43	.59	.79	.88	.40	.63	.40	.62	.61	.68	.53	.69
cPNN	.42	.61	.82	.90	.61	.73	.65	.75	.69	.72	.80	.82
mcLSTM	.33	.51	.67	.84	.33	.49	.41	.61	.25	.53	.43	.60
	 (M)		 (S)		 (M)		F_{1+} (M)		F_{2-} (S)		F_{1-} (M)	
cLSTM	.66	.86	.45	.62	.81	.90	.60	.67	.49	.65	.60	.68
cPNN	.71	.85	.58	.69	.85	.91	.57	.64	.74	.77	.66	.67
mcLSTM	.71	.85	.33	.49	.68	.84	.26	.50	.47	.62	.32	.56
	 (S)		 (M)		 (S)		F_{1-} (S)		F_{2-} (M)		F_{1+} (S)	
cLSTM	.66	.86	.44	.63	.69	.85	.23	.48	.73	.77	.39	.60
cPNN	.73	.85	.59	.69	.86	.90	.57	.64	.74	.76	.65	.67
mcLSTM	.68	.84	.33	.50	.71	.86	.27	.50	.47	.62	.33	.55

accumulated data points, compute the Cohen’s Kappa score on them, and then update the model. We then average the scores from the first mini-batch after the last concept drift to b_t .

As we made in the cPNN paper [9], for each concept, we consider the performance after its first part (*start*) and after its end (*end*). *start* is represented by the first 50 mini-batches and is useful to evaluate how the models react to the drift. In the anytime scenario, we consider the corresponding number of data points ($50 * B$ where B is the mini-batch size). *end* allows us to evaluate how the models perform on the entire concept. All the experiments are executed ten times. To assess whether one model outperforms another, we conduct a statistical hypothesis test with $\alpha = 0.05$, considering the performance of the ten executions. We 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.

4 Results

Table 1 shows the comparison results between cPNN and its ablated versions in the periodic scenario. Since the models share the identical architecture on the first concept, we report the comparison from the second concept onwards. For each concept of each configuration, we report the mean of the Cohen’s Kappa scores achieved by the models on the 10 iterations after the first 50 mini-batches (*start*) and the last mini-batch (*end*). We highlight the statistically best-performing

Table 2: Average Cohen’s Kappa scores of the different models in the anytime scenario.

	SRWM								Weather							
	end	start	end	start	end	start	end	end	start	end	start	end	start	end	start	end
ARF	.36	.26	.33	.23	.36	.23	.33	.17	.48	.49	.19	.19	.48	.51		
ARF _T	.73	.74	.74	.73	.73	.74	.74	.30	.74	.76	.34	.33	.76	.76		
HAT	.35	.26	.34	.17	.34	.24	.33	.16	.38	.40	.15	.17	.42	.42		
HAT _T	.60	.63	.66	.63	.65	.60	.64	.29	.59	.59	.29	.30	.61	.59		
cPNN	.78	.33	.50	.69	.80	.47	.51	.50	.73	.77	.57	.57	.75	.79		
ARF																
ARF _T	.36	.21	.33	.28	.35	.20	.33	.18	.27	.43	.20	.20	.36	.47		
HAT	.35	.21	.32	.24	.35	.19	.33	.15	.28	.38	.18	.18	.37	.41		
HAT _T	.60	.57	.66	.63	.67	.64	.66	.29	.60	.58	.29	.30	.60	.58		
cPNN	.78	.33	.50	.64	.79	.43	.51	.50	.62	.71	.57	.62	.74	.76		
ARF																
ARF _T	.35	.23	.34	.23	.33	.23	.35	.47	.20	.21	.38	.46	.23	.21		
HAT	.74	.74	.74	.74	.74	.73	.74	.74	.30	.34	.76	.75	.38	.37		
HAT _T	.33	.26	.35	.19	.32	.24	.35	.38	.16	.17	.38	.39	.18	.17		
cPNN	.38	.58	.81	.47	.53	.69	.84	.54	.51	.60	.70	.71	.54	.63		
ARF																
ARF _T	.36	.17	.35	.23	.33	.21	.35	.47	.15	.19	.50	.49	.22	.21		
HAT	.74	.73	.74	.74	.74	.74	.74	.74	.30	.34	.76	.75	.38	.37		
HAT _T	.33	.18	.34	.23	.33	.19	.34	.38	.12	.16	.42	.40	.15	.16		
cPNN	.55	.58	.64	.63	.66	.62	.61	.58	.28	.31	.52	.56	.32	.32		
cPNN	.39	.57	.80	.45	.52	.54	.79	.54	.45	.47	.73	.76	.40	.50		

Table 3: Hypotheses tests results for SRWM and Weather. We indicate whether the hypothesis is verified (✓) or not (✗) for each of the four concepts, across every generator, configuration, and hypothesis. When a hypothesis does not apply to a specific concept we use “-”.

		H1	H2	H3	H4	H5	H6
SRWM		- - ✓ -	- - ✓ ✓	- ✓ - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✓ ✗ ✓ ✗
		- ✗ - ✓	- - ✓ ✓	- - - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✓ ✗ ✓ ✗
		- - ✓ -	- - ✓ ✓	- ✓ - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✗ ✗ ✓ ✗
		- ✓ - ✓	- - ✓ ✓	- - - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✗ ✗ ✓ ✓
Weather	$F_{1+} F_{2+} F_{1-} F_{2-}$	- - ✓ -	- - ✓ ✓	- ✓ - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓
	$F_{1+} F_{2-} F_{1-} F_{2+}$	- ✓ - ✓	- - ✓ ✓	- - - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✓ ✗ ✓ ✗
	$F_{2+} F_{1+} F_{2-} F_{1-}$	- - ✓ -	- - ✓ ✓	- ✓ - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✗ ✗ ✓ ✗
	$F_{2+} F_{1-} F_{2-} F_{1+}$	- ✓ - ✓	- - ✗ ✓	- - - -	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓	✗ ✗ ✓ ✓
Verified cases		11	15	4	32	32	19
Total cases		12	16	4	32	32	32

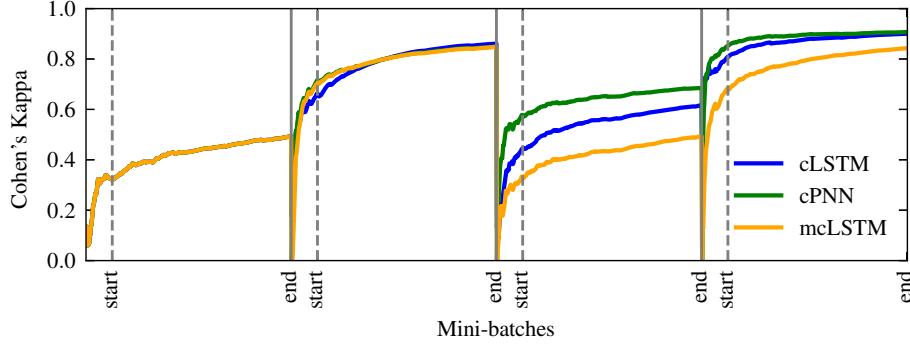
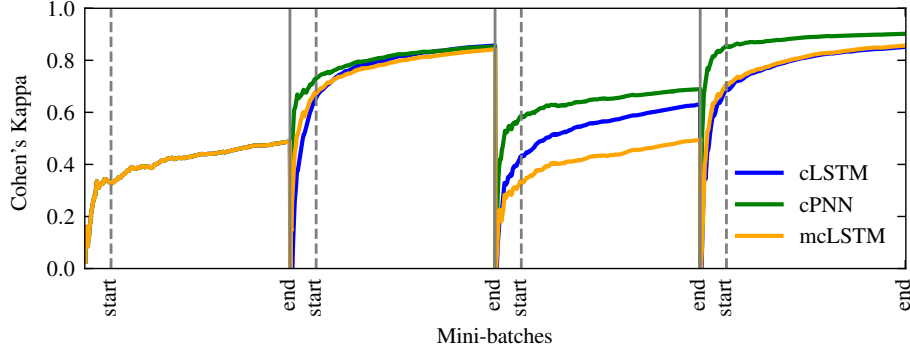
(a) Ablation study on SRWM  (periodic scenario).(b) Ablation study on SRWM  (periodic scenario).

Fig. 3: Ablation study’s average Cohen’s Kappa scores on two configurations of SRWM. Experiments are executed 10 times. Scores are reset after each drift. cPNN always outperforms the ablated versions in case of severe drift or when the new concept introduces an already-seen boundary function.

model in bold. Figures 3 and 4 report the complete average Cohen’s Kappa evolution on two configurations of SRWM and Weather. The x-axis represent the different mini-batches of the data stream. The values on the y-axis are the Cohen’s Kappa scores of the models considering the predictions on the mini-batches from the first following the last drift to the current one. Table 2 presents the results of the comparison between SML_C models, SML_T models, and the maximum performance achieved by cPNN or one of its ablated versions in the anytime scenario. We only present the maximum performance among the cPNN versions because the loss function doesn’t align with the inference objective in the anytime scenario. Consequently, the top-performing version of cPNN in the periodic scenario isn’t necessarily the optimal choice in the anytime scenario. This could result in applying an ensemble with cLSTM, cPNN, and mcLSTM. We also report the performance at the end of the first concept since it is an

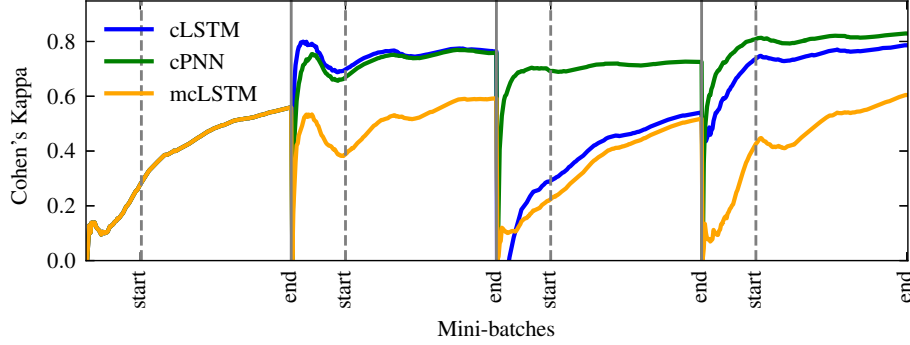
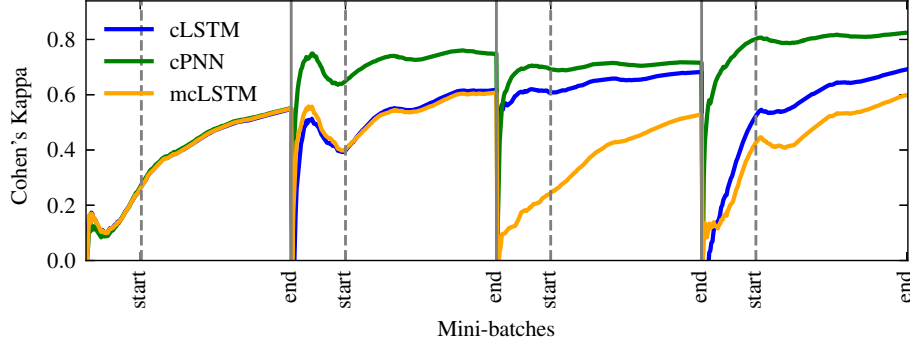
(a) Ablation study on Weather $F_{1+} F_{2+} F_{1-} F_{2-}$ (periodic scenario).(b) Ablation study on Weather $F_{1+} F_{2-} F_{1-} F_{2+}$ (periodic scenario).

Fig. 4: Ablation study’s average Cohen’s Kappa scores on two configurations of Weather. cPNN always outperforms the ablated versions in case of severe drift or when the new concept introduces an already-seen boundary function.

object of study. Figures 3 and 4 show the comparison between the maximum performance among the cPNN versions, SML_C , and SML_T . For simplicity, we only report ARF and ARF_T since they outperform HAT and HAT_T, respectively. The x-axis represents the single data points of the data stream. On the y-axis, we report the Cohen’s Kappa scores of the models considering the data points from the first following the last drift to the current one. Finally, Table 3 reports the summarization of our hypothesis tests for the four concepts of each configuration.

Concerning the ablation study, hypotheses **H1** and **H2** are quite always verified. This finding proves the effectiveness of cPNN in exploiting transfer learning to better adapt to drifts when the drift is severe or when the boundary function is known. The advantages of using cPNN in the case of severe drift are more evident in Weather configurations. This is because SRWM’s classification functions are easier to learn, and cLSTM and mcLSTM can react more quickly to the drift. Conversely, Weather contains more complex data from a realistic

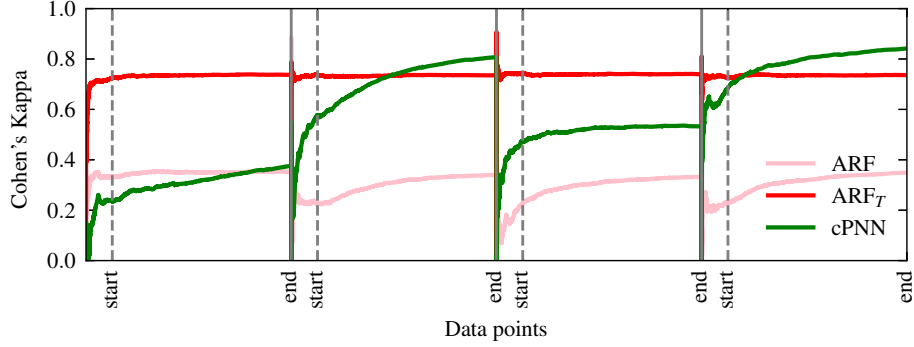
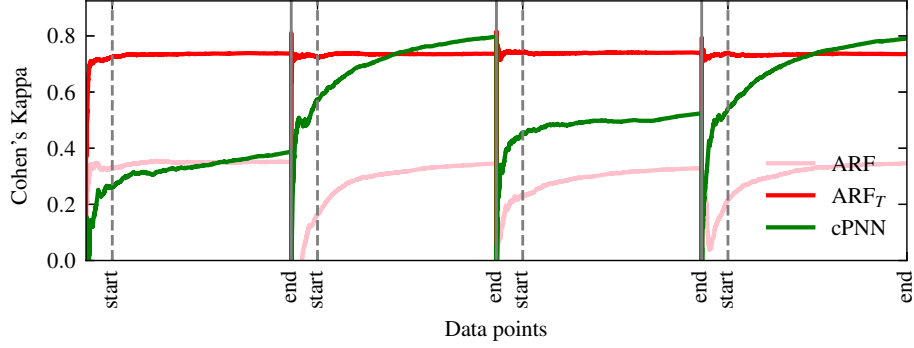
(a) SRWM $\blacksquare \blacktriangleleft \blacktriangleright \square$ (anytime scenario).(b) SRWM $\blacksquare \blacktriangleleft \blacktriangleright \square$ (anytime scenario).

Fig. 5: Average Cohen's Kappa scores on two configurations of SRWM in the anytime scenario. cPNN is computed as the maximum performance between cLSTM, cPNN, and mcLSTM. ARF cannot learn efficiently. The best-performing cPNN version can outperform ARF_T on the concepts associated with \square .

scenario. When the drift is mild, and the boundary function is new, cLSTM can start the concept with a higher performance than cPNN. These are the cases in which cLSTM capitalizes its previous optimal settings to optimize the new loss function. This finding allows us to confirm the hypothesis **H3**. Secondly, if the knowledge of the old concept is still valid, cPNN may slow down when calibrating its weights. However, at the end of the concept, cPNN outperforms cLSTM or obtains the same performance. Additionally, mcLSTM is usually the less-performing cPNN version. Using past knowledge is crucial to adapt to the new concept. cPNN does it with transfer learning. cLSTM does it by starting from the optimal setting of the previous concept. This could be enough only when the new concept is similar to the previous one.

When comparing cPNN and SML models, SML_C models cannot learn, reach poor performance, and struggle. Hypotheses **H4** and **H5** are, in fact, always

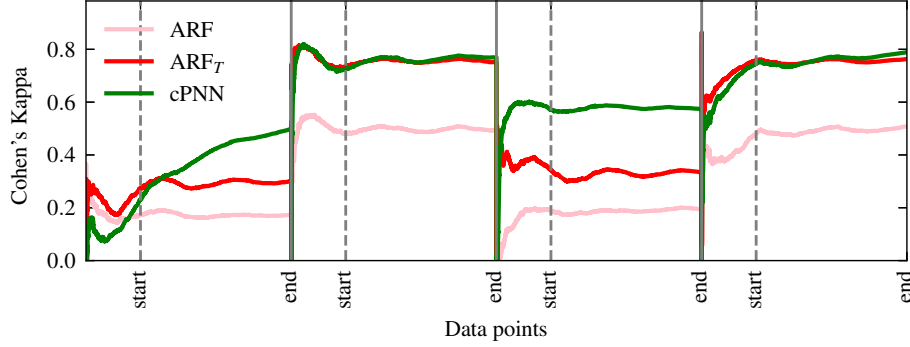
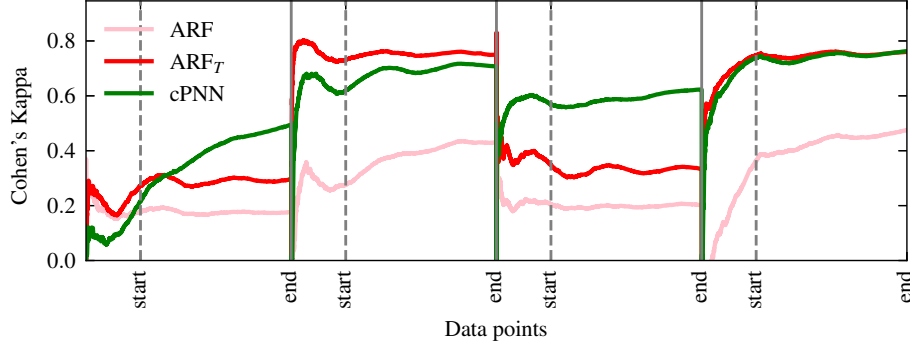
(a) Weather F_{1+} F_{2+} F_{1-} F_{2-} (anytime scenario).(b) Weather F_{1+} F_{2-} F_{1-} F_{2+} (anytime scenario).

Fig. 6: Average Cohen’s Kappa scores on two configurations of Weather in the anytime scenario. cPNN is computed as the maximum performance between cLSTM, cPNN, and mcLSTM. ARF cannot learn efficiently. The best-performing cPNN version outperforms ARF_T on F_1 .

verified, and SML_C models are always outperformed by at least its version with TA and one cPNN model. Temporal augmentation is crucial to improve the SML models’ performance. **H6** is verified in most cases. On SRWM, the best-performing cPNN version can outperform the best-performing SML_T model on the concepts associated with \square . On the most complex function (\sqsubseteq), the best-performing cPNN version cannot reach the performance of SML_T but constantly outperforms SML_C models without TA. When we move to a more realistic approach like Weather, the best-performing cPNN version outperforms SML_T models on F_1 .

One key finding is that SML_T models perform poorly on the only boundary function that reduces the number of data points with label changes compared to the previous. F_1 , in fact, retains the previous label for only 68% of data points, resulting in SML_T models achieving Cohen’s Kappa scores lower than 0.4 for related concepts. Conversely, F_2 and the SRWM functions keep the previous

label for nearly 90% of data points. ARF_T achieves scores exceeding 0.72 on these functions. This suggests that TA may be biased on the previous label. This suggestion is confirmed by Figure 5, where ARF_T performance on SRWM keeps unaltered over the entire data stream. On Weather (Figure 6), instead, it takes longer to converge to the final score, and the drift has a clear impact. Additionally, when the label y_t of data point d_t is unavailable before receiving X_{t+1} , TA cannot be applied since it relies on the previous labels.

5 Conclusion

This study extensively analyzed various streaming models' behaviour in handling concept drifts with complex temporal dependence. The examined models included classical SML, SML with TA, and cPNN. We initially used a synthetic data generator to simulate temporal dependence, then transitioned to real data for a more realistic scenario. An ablation study compared cPNN's response to drift with SGD, showing cPNN's adeptness, particularly in severe drift or previously encountered functions. Comparisons with classical SML models highlighted cPNN's efficiency in managing temporal dependence, outperforming SML and SML with TA models. Additionally, Classical SML models were not able to learn efficiently.

Two main limitations of cPNN are its complexity grows linearly with the number of concepts and the difficulty of hyperparameter selection in data streams. Since assuming the exact timestamps of drifts is unrealistic, future research could explore using drift detection methods. Additionally, cPNN is a periodic learner requiring mini-batches. We proposed a first approach to transform it into an anytime classifier. Modifying the loss function could enhance classification at any time. Since our results suggest TA may be biased toward the previous label, a future study can investigate TA's behaviour with complex functions that frequently change labels. Additionally, in the case of delayed labels TA cannot be applied and other methods should be found to tame temporal dependence. Finally, while cPNN is naturally robust to forgetting, we leave the study of forgetting in cLSTM and SML models for future research.

References

1. Bifet, A., Gavaldà, R.: Learning from Time-Changing Data with Adaptive Windowing. In: SDM. pp. 443–448. SIAM (2007)
2. Bifet, A., Gavaldà, R.: Adaptive Learning from Evolving Data Streams. In: IDA. LNCS, vol. 5772, pp. 249–260. Springer (2009)
3. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine learning for data streams: with practical examples in MOA. MIT press (2018)
4. Bifet, A., Read, J., Zliobaite, I., Pfahringer, B., Holmes, G.: Pitfalls in Benchmarking Data Stream Classification and How to Avoid Them. In: ECML/PKDD (1). Lecture Notes in Computer Science, vol. 8188, pp. 465–479. Springer (2013)
5. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time series analysis: forecasting and control. John Wiley & Sons (2015)

6. Cossu, A., Carta, A., Bacciu, D.: Continual Learning with Gated Incremental Memories for sequential data processing. In: IJCNN. pp. 1–8. IEEE (2020)
7. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with Drift Detection. In: SBIA. LNCS, vol. 3171, pp. 286–295. Springer (2004)
8. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: KDD. pp. 329–338. ACM (2009)
9. Giannini, F., Ziffer, G., Della Valle, E.: cPNN: Continuous Progressive Neural Networks for Evolving Streaming Time Series. In: PAKDD (4). Lecture Notes in Computer Science, vol. 13938, pp. 328–340. Springer (2023)
10. Godsey, S.E., et al.: Eleven years of mountain weather, snow, soil moisture and streamflow data from the rain–snow transition zone—the Johnston Draw catchment, Reynolds Creek Experimental Watershed and Critical Zone Observatory, USA. *Earth System Science Data* **10**(3), 1207–1216 (2018)
11. Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfahringer, B., Holmes, G., Abdessalem, T.: Adaptive random forests for evolving data stream classification. *Mach. Learn.* **106**(9–10), 1469–1495 (2017)
12. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016)
13. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
14. 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)
15. 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)
16. 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)
17. 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)
18. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: M5 accuracy competition: Results, findings, and conclusions. *IJOF* **38**(4), 1346–1364 (2022)
19. 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)
20. McHugh, M.L.: Interrater reliability: the kappa statistic. *Biochemia medica* **22**(3), 276–282 (2012)
21. Pearson, K.: The problem of the random walk. *Nature* **72**(1865), 294–294 (1905)
22. Read, J., Zliobaite, I.: Learning from Data Streams: An Overview and Update. *CoRR abs/2212.14720* (2022)
23. 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)
24. 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)
25. 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