Chaotic Continual Learning

Touraj Laleh [*1 2] **Mojtaba Faramarzi** [*1 2] **Irina Rish** [1 2 3] **Sarath Chandar** [1 4 3]

Abstract

Training a deep neural network model usually requires multiple iterations, or epochs, over the training data set, in order to better estimate the parameters of the model. However, in continual learning, this process results in catastrophic forgetting which is one of the core issues of this domain. Most proposed approaches for this issue try to compensate for the effects of parameter updates in the batch incremental setup in which the training model visits a lot of samples for several epochs. However, it is not realistic to expect training data will always be fed to model in a batch incremental setup. This paper proposes a chaotic stream learner that mimics the chaotic behavior of biological neurons and does not updates network parameters. In addition, it can work with fewer samples compared to deep learning models on stream learning setup. Our experiments on MNIST, CIFAR10, and Omniglot show that the chaotic stream learner has less catastrophic forgetting by its nature in comparison to a CNN model in continual learning.

1. Introduction

Continual learning assumes that a learning agent is presented with a sequence of different "tasks" (i.e., data coming from different probability distributions), where each task is a sequence of experiences from the same distribution (Riemer et al., 2018). The human brain can continuously learn different tasks without any of them negatively interfering with each other. However, learning a set of sequential tasks in the neural networks degrades the performance of the models. This is one of the biggest challenge in continual learning which is known as catastrophic forgetting (McCloskey & Cohen, 1989; Chen & Liu, 2018).

[*]Equal contribution [1]Mila - Quebec AI Institute [2]University of Montreal [3]Canada CIFAR AI Chair [4]École Polytechnique de Montréal. Correspondence to: Touraj Laleh <tourajla@mila.quebec>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

Most approaches have been proposed to alleviate the catastrophic forgetting are categorized into one of three main categories, including replay-based, regularization-based, and parameters isolation based methods according to the task-specific information that is stored and used through sequential learning process (De Lange et al., 2019). Replay based methods store exemplars in the replay buffer to alleviate the catastrophic forgetting while learning new tasks (Rebuffi et al., 2017; Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018). Since it is not always possible to keep exemplars, regularization-based methods propose extra regularization to consolidate previous knowledge when learning new tasks (Kirkpatrick et al., 2017). In the third approach, the capacity of the model is not restricted and the model architecture can be expanded (Xu & Zhu, 2018; Rusu et al., 2016), copied (Aljundi et al., 2017) or frozen to alleviate catastrophic forgetting. Some of the solutions in this approach mask out the parameters or even neurons that are used for the previous tasks (Mallya & Lazebnik, 2018; Fernando et al., 2017).

Backpropagation is the main reason for catastrophic forgetting in continual learning and most proposed approaches alleviate this issue by reducing the negative effects of backpropagation. In addition, in most real world scenarios a learning agent receives a very limited number of training samples in each task similar to a few-shot learning setup (Antoniou et al., 2020). However, most proposed approaches in continual learning needs to revisit training data for several epochs while learning a new task. Inspired by the chaotic firing behavior of biological neurons many approaches have been proposed to avoid backpropagation. ChaosNet is one of those approaches that proposed a 1D chaotic dynamic using the Generalized Lurth Series (GLS) as a chaotic neuron (Balakrishnan et al., 2019). However, ChaosNet can not compete with the deep neural network model in an image classification task since a deep learning model can visit training samples in several epochs. Adapting the GLS neuron in the continual few-shot learning and stream learning can be considered as an alternative approach since it suffers less catastrophic forgetting. Following the ChoasNet, we used a GLS neuron to simulate the chaotic behavior of a biological neuron to encode images with chaotic dynamics. We propose a GLS stream learner that uses a linear 1D ChaosNet neuron as a continual learner component. We also propose

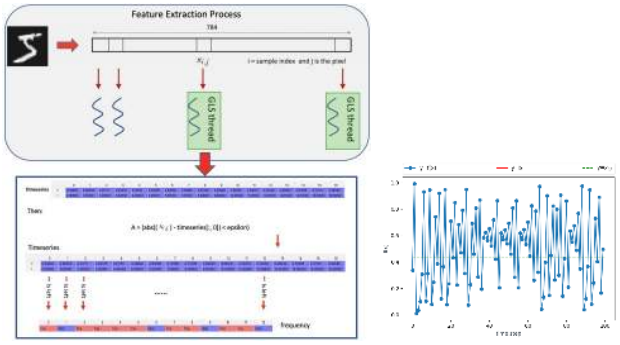


Figure 1. The chaotic GlS based stream learner process (left) and the feature extraction summary for a pixel, $x_{i,j}$, using skew-tent dynamic mapping (right).

using some chaotic transformation as a data augmentation technique. Our results demonstrate that our chaotic learner has noticeable results in comparison to a CNN model in the batch incremental and stream learning setups for the image class incremental classification tasks.

2. GLS Stream Learner

In the human brain, dendrites are exquisitely specialized cellular compartments that critically influence how neurons collect and process information. Retinal ganglion cell (RGC) dendrites receive synaptic inputs from bipolar and amacrine cells, thus allowing cell-to-cell communication and flow of visual information (Agostinone & Di Polo, 2015). Following ChaosNet, GLS Stream Learner uses Generalized Lurth Series (GLS) as a time-series that is defined as $f_{t+1} = f_t(q)$ where q initializes the dynamic. The Generalized Lurth Series is used to simulate the firing and not firing behavior of a biological neuron (Balakrishnan et al., 2019). To compute GLS series, we use the following mapping:

$$f(x) = \begin{cases} \frac{x}{b} & 0 \leq x < b \\ \frac{a(x-c)}{(1-b)} & b \leq x < 1 \end{cases} \tag{1}$$

where b controls the shape of the attractors. Setting $(c = 1, a = -1)$ gives a tent shape attractors known as Skew-Tent ($f_{Skew-Tent}$). And, the mapping with $(c = b, a = 1)$ creates an attractors in the shape of two separated lines known as Skew-Binary ($f_{Skew-Binary}$). Skew-tent and skew-binary have been used for encryption of information in many domains (Li et al., 2019). Appendix A illustrates the output of Skew-tent and skew-binary through the time that simulates either firing or not firing responses of a neuron corresponding to inputs.

2.1. Feature Extraction

GLS learner uses normalized images. So, the pixels are scalar numbers in $[0, 1]$. The feature extractor process runs several threads. And, each thread encodes the pixel j from x_i to a probability of firing rate count denoted as a $P_{i,j}$. The feature extraction process creates the GLS time-series with either skew-tent or skew-binary mapping. Then, the GLS thread tries to find the first ϵ-neighborhood point in the time-series to the pixel information. Then, it computes the firing rate count for each pixel as follow:

$$P_{i,j} = \frac{\text{False count}}{\text{length of frequency list}}, \tag{2}$$

The result is the decoded information vector P for x_i. Since we encode each pixel to a probability number, the size of P will be the same as image size. Figure 1 shows the feature extraction process in detail.

2.2. Training

At time T, set of samples of new classes arrive in either batch incremental fashion or as a stream of data. In this case, we have $D_T \subseteq D_{train}$ where T is the task time for N samples from either i.i.d or non i.i.d observations of $(x_i,\ c_i)$ pairs where $c_i \in C$. And, C is the set of classes that the stream learner should learn at time T. For, each sample x_i, the stream learner creates the vector P_i that is the extracted feature from x_i. Then, the stream learner computes the mean representation, $M_{c_i}^{SL}$, for m samples of class c_i that have seen at time T as follow:

$$M_{c_i}^{SL} = \frac{1}{m}\Big[\sum_{i=1}^{m} P_{i1}^{c_i}, \sum_{i=1}^{m} P_{i2}^{c_i}, \ldots, \sum_{i=1}^{m} P_{in}^{c_i}\Big], \tag{3}$$

The batch incremental learner visits samples in several epochs. Therefore, GLS learner should compute a new vector, $M_{c_i}^{b_j}$, for the epoch j and appends it to the batch incremental representatives as follow:

$$M_{c_i}^{BSL} = \Big[M_{c_i}^{b_1}, M_{c_i}^{b_2}, \ldots, M_{c_i}^{b_n}\Big], \tag{4}$$

where n is the number of epochs and $M_{c_i}^{b_j}$ is computed for the epoch j using 3.

Replay Buffer: GLS stream learner keeps either $M_{c_i}^{BSL}$ or $M_{c_i}^{SL}$ for the batch incremental or stream learning setup, respectively, in the replay buffer instead of keeping exemplars in the buffer.

2.3. Classification

Let assume $\varphi(x)$ extracts features for a sample $x \in X_{test}$ as described in 2.1. Then, GLS stream learner predicts the target as follow:

$$y^* = \underset{y=1,\ldots,t}{\operatorname{argmin}} \|\varphi(x) - M_c\|, \tag{5}$$

Where $c \in C$ (all classes have learned so far) and $\|\varphi(x) - M_c\|$ is the cosine similarity distance of two $\varphi(x)$

Figure 2. Batch Incremental (left) v.s. Stream Learning (right).

and M_c vectors that can be either either $M_{c_i}^{BSL}$ or $M_{c_i}^{SL}$ for the batch incremental or stream learning, respectively.

2.4. Chaotic Data Augmentation

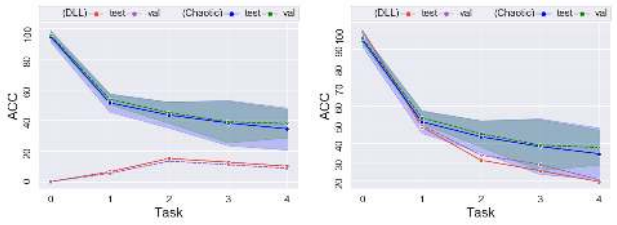
Chaotic data augmentations can be applied on the train set to improve the performance of the continual learner. In this work, we proposed using "baker's" transformation that resembles the process of repeatedly stretching a piece of dough and folding it in two. Equation 6 depicts "baker's" mapping where i and j are the row and column of each pixel in an image.

$$f_{baker's}(x_i, x_j) = \begin{cases} (2x_i, \lambda x_j) & (0 \leqslant x_i \leqslant \frac{1}{2}) \\ (2x_i - 1, \lambda x_j + \frac{1}{2}) & (\frac{1}{2} < x_i \leqslant 1) \end{cases} \quad (6)$$

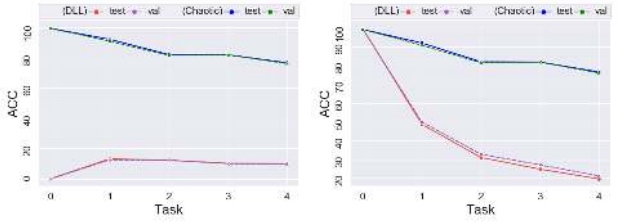
The dynamics of Hnon map may be decomposed into an area-preserving bend, followed by a contraction, followed by a reflection in the line $y = x$. Hnon map $f : \mathbb{R}^2 \to \mathbb{R}^2$ as $f_{Hnon}(x_i, x_j) = \left(x_j + 1 - a x_i^2, b x_i\right)$. The solenoid dynamic mapping also can be used as a rotation transformation (Falconer, 2004).

3. Experiments

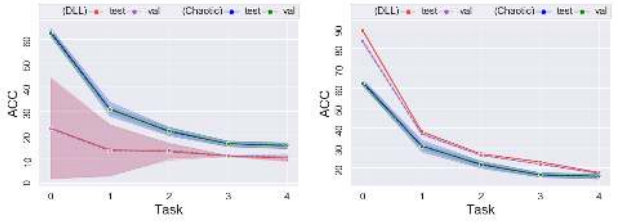
There are three different data paradigms of stream learning for evaluating continual learning models based on the way the training data is organized (Hayes et al., 2018; Antoniou et al., 2020). The model visits a limited number of samples in only one epoch in either one of the following setups in the stream learning context. In the first setup, the data stream is completely unordered. In the second setup, the data stream is ordered by the class and the models learns classes incrementally which results in catastrophic forgetting. In the third setup, data is organized on batches from specific instances of categories that can be revisited (Antoniou et al., 2020). diversely, the model incrementally learns new classes and it is allowed to revisits samples for several epochs while training a new task in the batch incremental setup. To evaluate our approach we follow the second paradigm that is more aligned with real scenarios. Figure 2 shows the comparison of batch incremental learning and the stream learning setups.



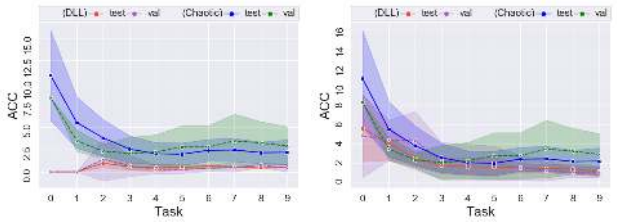
(a) Permuted-MNIST divided into 5 tasks with 2 classes per task.



(b) MNIST divided into 5 tasks with 2 classes per task.



(c) CIFAR-10 divided into 5 tasks with 2 classes per task.



(d) Omniglot divided into 10 tasks with 20 classes per task.

Figure 3. Performance comparison of the chaotic learner v.s. the CNN model in the stream of data (left) and Chaotic learner performance comparison with the CNN model in a batch incremental learning setup (right). We repeated the experiment three times to report except for the MNIST dataset that has the significant marginal performance. The higher number is better.

3.1. Experiment Setup

We compared the performance of GLS stream learner with a convolutional neural network (CNN) using three different datasets including MNIST (LeCun & Cortes, 2010), CIFAR-10 (Krizhevsky, 2009), Omniglot (Lake et al., 2015) and Permuted-MNIST which is the same data as MNIST dataset when every pixel is randomly permuted. Appendix B explains the architecture of a CNN model that is used

in our experiments. In addition, we have two different experiment setups for each dataset. The first one is a batch incremental setup for a CNN model that revisits samples for 50 epochs for each task. The second one is a stream learning setup for the chaotic learner that uses a few samples to train the model for each task. For each dataset, data is divided into different tasks and each task has a certain number of classes. We run three experiments for each dataset including batch incremental for CNN model, stream setup for GLS learner, and evaluating the CNN model performance trained in a few-shot manner in the stream learning setup. In our experiment, we observed that Skew-binary mapping achieves 3% more accuracy performance at each task in comparison to the Skew-tent. Therefore, all reported results for GLS stream learner are based on Skew-binary mapping with $\{b = 0.331, \epsilon = 0.01, q = 0.336, \text{time step} = 20000\}$ for MNIST, Permuted-MNIST and Omniglot datasets And $\{b = 0.331, \epsilon = 0.008, q = 0.136, \text{time step} = 30000\}$ for CIFAR-10.

MNIST and Permuted-MNIST: We split the MNIST and Permuted-MNIST samples into 5 tasks with two classes per task. In the stream setup, we train models with 3 batches and 32 samples per batch. Figures 3(b) and 3(a) illustrate the comparison results for MNIST and Permuted-MNIST. They show chaotic learner has significant marginal performance in comparison to the CNN model in both setups.

CIFAR-10: Figures 3(c) illustrates the comparison results on CIFAR-10. CIFAR-10 training data is split into the 5 tasks with two classes per each task. For stream learner setup, we train models with 4 batches and 64 samples per batch. Unlike MNIST, the CNN model has better performance (figures 3(c) right) in batch incremental learning setup on CIFAR-10. However, the chaotic learner still lead to better performance in the stream setup approach.

Omniglot: To evaluate the performance on the Omniglot dataset (Lake et al., 2015), we designed two experimental setups. From 964 classes in the background TRUE set of the Omniglot dataset, we only chose 200 classes for this experiment. We split the selected data into 10, 20, and 40 tasks such that each task includes 20, 10, and 5 classes, respectively. We have 20 samples per class in this setup. We selected 60 percent of samples for training. Therefore, we have a few samples for training in the batch incremental and stream learning setups (12 samples per class in the training set). Figure 3(d) represents the result of the Omniglot dataset that is divided into 10 tasks. Appendix C contains further comparison results on Omniglot for the 20 and 40 tasks. The chaotic stream learner shows better results compared to the CNN model in both batch incremental and stream learner setups on Omniglot.

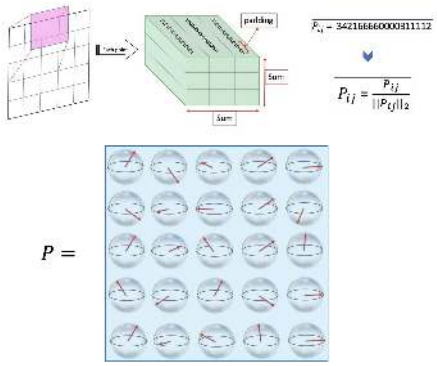


Figure 4. The process of computing a higher-level abstraction for each pixel using normalized correlated extracted features.

4. Future Work

To preserve the correlation between the extracted features, we can use a moving feature extractor block on the sample x_i, and calculate the correlated features for each pixel that lies in the moving block. Figure 4 briefly shows the process. First, we extract features associated with each pixel using the GLS feature extractor described in 2.1 for all pixels in the block. Adding a zero padding to the extracted firing rates helps to have the same size firing rate series. Applying an element-wise sum and then normalize the vector of the firing rates gives a higher-level abstraction of the correlated feature extracted for the pixel, P_{ij}, that is defined as follow:

$$P_{ij} = \frac{P_{ij}}{\|P_{ij}\|_2}, \tag{7}$$

All extracted P_{ij} have the same size but different angels. The angels can represent the higher-level abstraction of the correlated feature associated with each pixel and its neighbors. Our next step in this direction is experimenting with the effectiveness of using higher-level correlated abstraction instead of considering a standalone extracted futures using the GLS feature extractor in the stream learning setup.

5. Conclusion

In this work, GLS stream learner is proposed as a novel approach to alleviate the catastrophic forgetting in the context of continual few-shot learning. This approach provides a mechanism based on the chaotic structure of a biological neuron that provides a different perspective from the most continual learning approaches. According to our experiment, this single chaotic neuron causes less forgetting in comparison to a deep learning model that needs a lot of time to train and more parameters to learn in batch incremental and stream learning setups. The deep learning model achieved profound performance in the image classification task. However, it suffers from catastrophic forgetting prob-

lems because of the backpropagation mechanism to update their parameters in the continual learning context. GLS stream learner can show the importance of thinking to find an alternative solution that more suitable for stream and continual few-shot learning setups.

References

Agostinone, J. and Di Polo, A. Retinal ganglion cell dendrite pathology and synapse loss: Implications for glaucoma. In *Progress in brain research*, volume 220, pp. 199–216. Elsevier, 2015.

Aljundi, R., Chakravarty, P., and Tuytelaars, T. Expert gate: Lifelong learning with a network of experts. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.753. URL http://dx.doi.org/10.1109/CVPR.2017.753.

Antoniou, A., Patacchiola, M., Ochal, M., and Storkey, A. Defining benchmarks for continual few-shot learning. *arXiv preprint arXiv:2004.11967*, 2020.

Balakrishnan, H. N., Kathpalia, A., Saha, S., and Nagaraj, N. Chaosnet: A chaos based artificial neural network architecture for classification. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(11):113125, Nov 2019. ISSN 1089-7682. doi: 10.1063/1.5120831. URL http://dx.doi.org/10.1063/1.5120831.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Chen, Z. and Liu, B. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

Falconer, K. *Fractal geometry: mathematical foundations and applications*. John Wiley & Sons, 2004.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017. URL http://arxiv.org/abs/1701.08734.

Hayes, T. L., Kemker, R., Cahill, N. D., and Kanan, C. New metrics and experimental paradigms for continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2031–2034, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proc. of the national academy of sciences*, 2017. URL https://www.pnas.org/content/pnas/114/13/3521.full.pdf.

Krizhevsky, A. Learning multiple layers of features from tiny images. 2009.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Li, C., Zhang, Y., and Xie, E. Y. When an attacker meets a cipher-image in 2018: A year in review, 2019.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017. URL https://arxiv.org/abs/1611.07725.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018. URL https://arxiv.org/abs/1810.11910.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks, 2016.

Xu, J. and Zhu, Z. Reinforced continual learning, 2018.

Appendices

A. Skew-Tent and Skew-Binary

Figure 5 shows ten steps movement in the time-series using skew-binary (left) and skew-tent (right). Repeating the process for several steps shows that the tent shape attractor or two separated lines attractor for skew-tent and skew-binary, respectively. Red and blue colors in scatter plots illustrated in figure 6 represents the active and passive status of a dendrite. It mimics either firing or not firing the response of a neuron corresponding to the input. Figure 7 shows this behavior through time. All points above the red line, where is defined based on the "b" hyperparameter, can be considered as true (firing) and the bellow points as false (not firing).

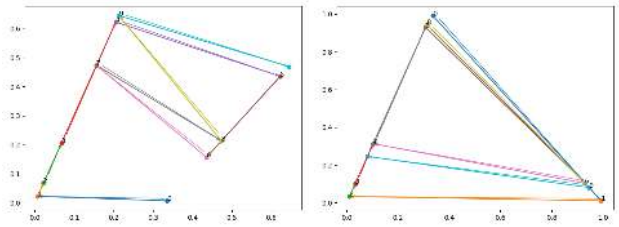


Figure 5. The skew-binary (left) and skew-tent (right) movement steps after applying ten-time steps in the time-series.

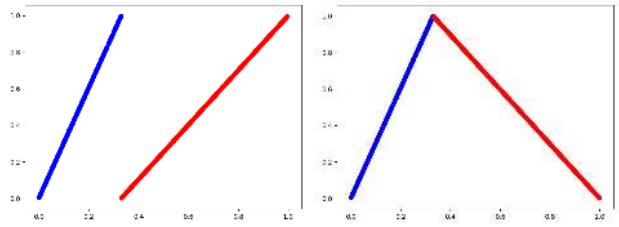


Figure 6. The Skew-Binary (left) and Skew-Tent (right) at tractors after 1000 time steps.

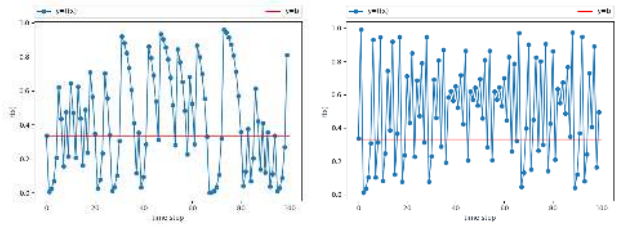


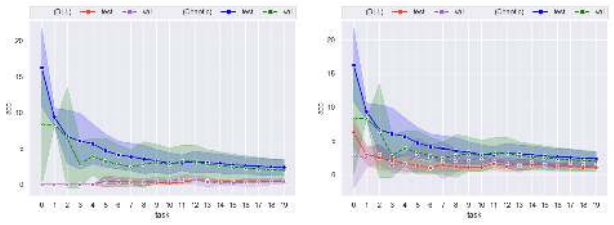
Figure 7. The Skew-Binary (left) and Skew-Tent (right) outputs for 100 time steps through time.

B. CNN Model Architecture

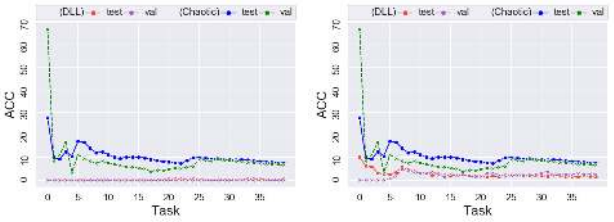
The CNN model that used in the experiments has the following architecture. It has 4 convolutional and 4 fully connected layers. In addition, the convolutional layers have 3, 10, 20 and 40 inputs and 10, 20, 40 and 64 output channels with 5, 5, 3, and 5 kernel size with stride of 1, respectively. The feature extractor part is followed by two fully connected layers that contain 680 and 280 neurons followed by a softmax module.

C. Omniglot Result

Figure 8 shows the further comparison on the Omniglot dataset with 20 and 40 tasks setup with 10 and 5 classes per task, respectively.



(a) Omniglot divided into 20 tasks with 10 classes per task.



(b) Omniglot divided into 40 tasks and 5 classes per task.

Figure 8. Performance comparison of the chaotic learner v.s. the CNN model in the stream of data (left) and Chaotic learner performance comparison with the CNN model in a batch incremental learning setup (right). The higher number is better.