Preprint of paper from ICASSP 2021.

UNSUPERVISED CLUSTERING OF TIME SERIES SIGNALS USING NEUROMORPHIC ENERGY-EFFICIENT TEMPORAL NEURAL NETWORKS

Shreyas Chaudhari *Harideep Nair* *José M.F. Moura* *John Paul Shen*

Carnegie Mellon University - Electrical and Computer Engineering Department

ABSTRACT

Unsupervised time series clustering is a challenging problem with diverse industrial applications such as anomaly detection, bio-wearables, etc. These applications typically involve small, low-power devices on the edge that collect and process real-time sensory signals. State-of-the-art time-series clustering methods perform some form of loss minimization that is extremely computationally intensive from the perspective of edge devices. In this work, we propose a neuromorphic approach to unsupervised time series clustering based on Temporal Neural Networks that is capable of ultra low-power, continuous online learning. We demonstrate its clustering performance on a subset of UCR Time Series Archive datasets. Our results show that the proposed approach either outperforms or performs similarly to most of the existing algorithms while being far more amenable for efficient hardware implementation. Our hardware assessment analysis shows that in 7 nm CMOS the proposed architecture, on average, consumes only about 0.005 mm^2 die area and 22 μW power and can process each signal with about 5 ns latency.

Index Terms— Temporal Neural Networks, Unsupervised Learning, Time-Series Clustering, Neuromorphic Chip

1. INTRODUCTION

Supervised deep neural networks have revolutionized the field of Machine Learning and achieve state-of-the-art results on many applications such as Computer Vision, Natural Language Processing, etc. However, they require 1) extensive amounts of labeled data, 2) large power-hungry compute resources, and 3) separate training and testing phases leading to poor generalization across unseen data patterns. These constraints become particularly pertinent for industrial *edge* applications involving time-series signals such as anomaly detection, bio-wearables, smart city sensors, etc. Unsupervised time-series clustering, which groups signals according to similar features, overcomes the first limitation. The second and third limitations raise the need for online processing that can dynamically capture the changes in *macro* patterns over time, within a tiny power budget.

Several techniques for time-series clustering have been studied in the literature. The work in [1] proposes a deep learning based approach to learn feature representations for enhanced clustering performance. They propose an encoder-decoder architecture and train the model using a joint classification, reconstruction and K-means loss function. While their approach achieves state-of-the-art performance, it requires the deep network to be trained a-priori using synthetic data. Furthermore, the approach, involving multi-layer RNNs and backpropagation, is not suitable for computationally and memory constrained edge devices. A method for learning shapelets, or discriminative segments of a time series, in an unsupervised manner has been proposed in [2]. While typical shapelet based methods exhaustively search a set of candidates to find the optimal shapelets for clustering, the USSL algorithm described in [2] builds on [3] to formulate the shapelet learning problem as an optimization problem that can be solved using coordinate descent. However, the authors use pseudo-labels to iteratively improve the quality of the discovered shapelets. While there are many other such works on unsupervised clustering in literature, low-power online learning algorithms have not been widely explored.

In this paper we present a low-power, online, unsupervised processor for time-series clustering based on the recently proposed Temporal Neural Networks (TNNs). TNNs are a specific class of Spiking Neural Networks (SNNs) that encode and process information *temporally* via relative spike timing relationships and are defined by a rigorous space-time algebra [4, 5]. The space-time computing framework of TNNs resemble that of the mammalian neocortex and can be directly implemented using off-the-shelf CMOS technology [6]. To the best of our knowledge, this is the first work to propose a TNN-based approach for time-series clustering.

2. BACKGROUND

2.1. SNNs and TNNs

Biologically inspired spiking neural networks have become increasingly popular due to their ability to perform pattern recognition tasks with ultra-low power consumption. These networks are typically composed of an encoding layer, one or more fully connected layers, and most recently, convolution layers [7, 8, 9, 10, 11, 12]. Most spiking neural networks in the literature either encode the input stimulus as the spiking

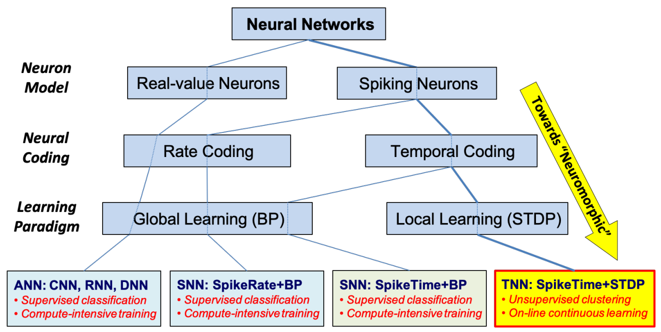

Fig. 1. Neural Network Taxonomy [6]

rate and/or use backpropagation for learning (Figure 1), both of which are not biologically plausible. Temporal neural networks (TNNs) employ intensity-to-latency temporal encoding and Spike Timing Dependent Plasticity (STDP) learning, both of which are biologically plausible. The architecture of TNNs is composed of a hierarchy of building blocks, namely, multi-synapse *neurons*, multi-neuron *columns* and multi-column *layers* and can be directly implemented in hardware using standard CMOS technology [6]. Synapses in TNN networks store weights and these weights are trained using STDP.

2.2. Temporal Encoding

Two neuron coding schemes have been primarily used in SNNs: *rate coding* and *temporal coding*. As implied by their name, rate-based coding schemes represent information using the frequency of spikes within a time interval and hence emit multiple spikes for a single input. Temporal coding methods, however, represent information in the relative time at which a spike occurs, rather than the spike frequency, leading to a single spike per input. Temporal codes are thus capable of accurately representing data with sparse spiking activity, and consume far less power than rate codes [5]. Furthermore, it has been experimentally shown in [13] that in some applications, encoding data with a population of neurons is more accurate than using a single neuron alone, i.e., it offers greater resolution for data representation. Hence, in this work, we apply both temporal and population coding mechanisms for representing time series signals as spike trains.

3. PROPOSED APPROACH

A block diagram of the proposed unsupervised time series clustering architecture is shown in Figure 2. The main components of our architecture are described in detail below.

3.1. Signal Encoding Layer

The encoding layer transforms the input time-series signal into a series of spike trains. Suppose the training dataset $\mathbf{D} \in \mathbb{R}^{N \times L}$ contains N uni-variate signals, where each signal

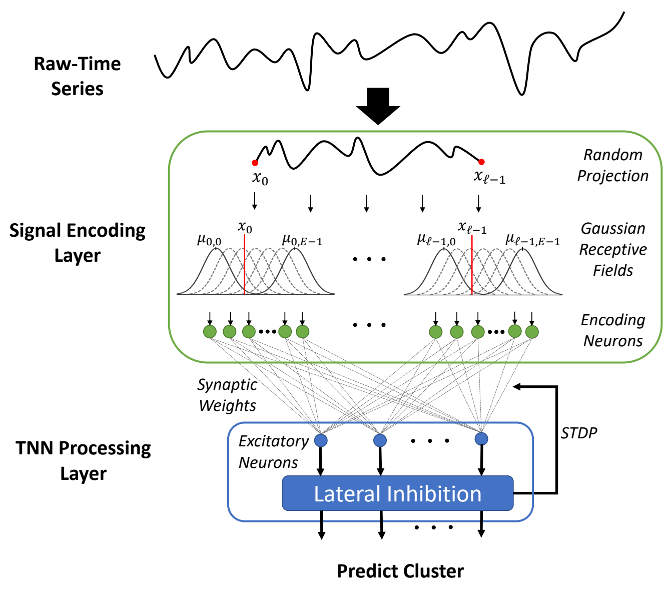

Fig. 2. Block diagram of the proposed TNN architecture

is of length L. We first perform dimensionality reduction on the incoming signals using sparse random projections. The Johnson-Lindenstrauss lemma [14] asserts that a set of points in n-dimensional Euclidean space can be randomly projected onto a p-dimensional linear subspace where $p < n$ such that the pairwise distances between points are roughly preserved. Achlioptis [15] showed that the transformation could be computed using a sparse matrix $\mathbf{P}$ whose ij-element can take one of the three values $\sqrt{3}$, 0, $-\sqrt{3}$ with respective probabilities $\frac{1}{6}$, $\frac{2}{3}$ and $\frac{1}{6}$. Thus we generate the reduced dataset $\widetilde{\mathbf{D}} = \mathbf{DP}$ where the projection matrix $\mathbf{P}$ has dimensions $L \times \ell$ (where $\ell << L$). The sparsity of $\mathbf{P}$ enables this lower dimensionality approach to be implemented on resource-constrained devices. It also significantly reduces the hardware implementation complexity of the TNN processing layer.

The transformed data $\widetilde{\mathbf{D}}$ is then converted to spike trains using Gaussian receptive fields resembling the approach in [16]. Specifically, let $\mathbf{x} = (x_0, x_1, \ldots, x_{\ell-1})$ be a data sample (row) from $\widetilde{\mathbf{D}}$. A point x_i is encoded by a population of E encoding neurons, each of which describes a Gaussian function. The parameters for the Gaussian functions for x_i are computed over all data samples (column i of $\widetilde{\mathbf{D}}$). The width σ_i and center μ_{ij} for the i^{th} time step and j^{th} neuron $(j = 0, 1, \ldots, E-1)$ are defined as:

$$\sigma_i = \frac{\gamma\,(x_{i,\,\max} - x_{i,\,\min})}{E-2}$$

$$\mu_{ij} = x_{i,\,\min} + \left(\frac{2j-3}{2}\right)\sigma_i$$

Here, $x_{i,\,\min}$ and $x_{i,\,\max}$ represent the minimum and maximum value in the i^{th} column of $\widetilde{\mathbf{D}}$ while γ is a scaling factor.

The time $t_j^{(e)}$ at which an encoding neuron j emits a spike for input x_i is:

$$f_j = \exp\left[-\frac{1}{2}\left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right]$$
$$t_j^{(e)} = \text{round}\left[T_{\max} \cdot (1 - f_j)\right]$$

Thus the encoding layer emits a total of $E \cdot \ell$ discrete spike trains for ℓ inputs. Here, $T_{\max}$ is an exclusive upper bound on the spike times generated by the encoding neurons. If $t_j^{(e)} = T_{\max}$, then no spike is emitted by the neuron. Note that our architecture model is *completely integer-based* and therefore highly amenable for efficient hardware implementation.

3.2. TNN Processing Layer

3.2.1. Excitatory Neurons

The fully connected TNN processing layer is composed of a single column of ramp-no-leak, integrate and fire neurons [6]. The number of neurons in the layer is equal to the number of clusters, while the number of synapses per neuron is equal to $E \cdot \ell$. The body (or membrane) potential of the k^{th} neuron in the TNN processing layer at time t is modeled as:

$$v_k(t) = \sum_j \rho\left(t - t_j^{(e)}, w_{kj}\right)$$

Here, w_{kj} is the synaptic weight from neuron j in the encoding layer to neuron k in the processing layer, and $t_j^{(e)}$ represents the input spike time from encoding neuron j. The ramp-no-leak response function, ρ, is defined as:

$$\rho(t, w) = \begin{cases} 0 & \text{if } t < 0 \\ t & \text{if } 0 \le t < w \\ w & \text{if } t \ge w \end{cases}$$

A neuron in the processing layer emits a spike immediately after its body potential exceeds some threshold θ. A neuron may only emit a single spike in a forward pass for any given data sample. After each forward pass (which includes lateral inhibition described below), the body potentials for all neurons in the layer are reset. The output spike time t_k of the k^{th} neuron in the processing layer is formally computed as:

$$t_k = \begin{cases} t_{out} & \text{if } \exists\, t_{out} \text{ s.t. } v_k(t_{out}) \ge \theta \text{ and} \\ & \nexists\, t < t_{out} \text{ s.t. } v_k(t) \ge \theta \\ T_{\max} & \text{otherwise} \end{cases}$$

3.2.2. Lateral Inhibition

Lateral inhibition is a biologically inspired technique in which an excited (i.e., spiking) neuron reduces the relative spiking activity of its neighboring neurons. In this work, we apply 1-Winner Take All (1-WTA) lateral inhibition to the output spike times of the processing layer. The earliest spike is allowed to propagate whereas all other spikes are suppressed. The output spiketime $t_k^{(o)}$ generated after 1-WTA mechanism can be formally computed as follows:

$$t_k^{(o)} = \begin{cases} t_k & \text{if } t_k = t_{\min},\ \nexists\, m < k \text{ s.t. } t_m = t_{\min} \\ T_{\max} & \text{otherwise} \end{cases}$$

Here, $t_{\min}$ is the earliest time at which any neuron in the processing layer fires. If multiple neurons spike at time $t_{\min}$, tie breaking selects the lowest index neuron. In case of no spike, the neuron with the maximum body potential is chosen.

3.3. STDP Training

A modified form of spike timing dependent plasticity (STDP) is used to train the synaptic weights of the TNN processing layer. In this work, we consider a stochastic, unsupervised learning rule with integer updates that is amenable to direct hardware implementation. The synaptic weight w_{kj} linking neuron j in the encoding layer to neuron k in the processing layer is updated according to the following rules:

Spike Time Conditions	$\boldsymbol{\Delta w_{kj}}$
$t_j^{(e)} \ne T_{\max},\ t_k^{(o)} = T_{\max}$	$+X_s$
$t_j^{(e)} \ne T_{\max},\ t_k^{(o)} \ne T_{max},$ $t_j^{(e)} \le t_k^{(o)}$	$+X_c \cdot \max(S_P(w_{kj}), X_{\min})$
$t_j^{(e)} \ne T_{\max},\ t_k^{(o)} \ne T_{\max},$ $t_j^{(e)} > t_k^{(o)}$	$-X_c \cdot \max(S_N(w_{kj}), X_{\min})$
$t_j^{(e)} = T_{\max},\ t_k^{(o)} \ne T_{\max}$	$-X_b \cdot \max(S_N(w_{kj}), X_{\min})$
$t_j^{(e)} = T_{\max},\ t_k^{(o)} = T_{\max}$	0

Table 1. STDP Update Rules [6]

Here, $X_s, X_c, X_b, X_{\min}$ are Bernoulli random variables with respective probabilities $\pi_s, \pi_c, \pi_b, \pi_{\min}$ of being one. Intuitively if the output spike time of the encoding neuron is less than or equal to that of the processing neuron, the corresponding synaptic weight is increased due to positive correlation. If the encoding spiketime is greater, the synaptic weight is reduced since the spike from the encoding neuron could not have led to a spike in the output neuron. After each STDP update, the stochastic weights are clamped within the interval $[0, w_{\max}]$. We restrict $\pi_s < \pi_c < \pi_b$ such that synaptic weights are conservatively increased. $S_P(\cdot)$ and $S_N(\cdot)$ are stabilizing random variables used to achieve a bimodal convergence of weights, and are characterized as:

$$\mathbb{P}\left[S_P(w_{kj}) = 1\right] = \left(\frac{w_{kj}}{w_{\max}}\right) \cdot \left(2 - \frac{w_{kj}}{w_{\max}}\right)$$
$$\mathbb{P}\left[S_N(w_{kj} = 1)\right] = \left(1 - \frac{w_{kj}}{w_{\max}}\right) \cdot \left(1 + \frac{w_{kj}}{w_{\max}}\right)$$

4. EXPERIMENTAL EVALUATION

4.1. Methodology

We evaluate our approach on 36 time-series datasets from the UCR time series archive [17], as done in [1], [2]. The datasets span a broad range of application domains including electrocardiogram (ECG) signals, human activity signals, image data converted to time series, food spectrographs and sensor recordings. The number of encoding neurons E per time series feature is equal to 8. We choose $\ell = \lfloor L/8 \rfloor$, where L is the signal length. Thus, the total number of inputs to the TNN is $E \cdot \ell \leq L$. The number of processing neurons is equal to the number of clusters. $T_{\max}$ and $w_{\max}$ are set to 16 and 7 respectively. *We use a 3-bit integer resolution for weights.*

The TNN is trained over multiple epochs for each dataset until approximate weight convergence is achieved. The cluster assignment is taken to be the index of the spiking neuron after 1-WTA. The winning neuron's spike time suggests the "confidence" in the cluster prediction. *Rand Index* (RI) is used as in [1] to measure clustering quality and is defined as:

$$\text{RI} = \frac{\alpha + \beta}{N(N-1)/2}$$

α (β) is the number of data sample pairs with same (different) label(s) that are assigned to same (different) cluster(s).

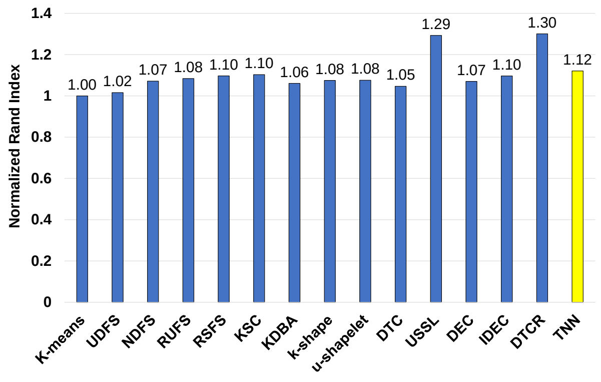

Fig. 3. Comparison of TNN vs. state-of-the-art methods using Rand Index normalized to K-means

4.2. Clustering Performance Results

The performance of TNN is compared to state-of-the-art algorithms reported in [1] [2]. Figure 3 depicts the rand index scores for all 15 algorithms (including TNN) averaged across the 36 datasets and normalized with respect to K-means. As shown in the figure, the proposed TNN approach outperforms 12 of the 14 algorithms even after the potential loss of information through random projections. The deep learning based approach DTCR [1] and shapelet based approach USSL [2] still outperform the TNN. However, DTCR involves high-dimensional floating point tensor processing and backpropagation. Meanwhile, USSL performs iterative coordinate descent in order to learn and refine shapelets according to pseudo labels. Both DTCR and USSL are far too computationally demanding for embedded edge devices.

USSL is also not ideal for continuous streaming data as shapelets are learned over the entire training dataset. Changes in *macro* patterns of the signal would require new shapelets to be explicitly learned. In comparison, our approach uses only 3-bit integer weights, is extremely hardware efficient (next subsection) and capable of continuously learning and adapting to *macro* level changes in the signal segments.

TNN Design	**Synapse Count**	**Area** [mm^2]	**Comp. Time** [ns]	**Power** [mW]
Largest	6750	0.033	6.50	0.155
Smallest	130	0.001	3.59	0.002
Average	970	0.005	5.07	0.022

Table 2. Hardware Complexity for proposed TNN approach

4.3. Hardware Complexity Analysis

We assess die area, power consumption and processing delay of the TNN processing layer, since the synapses in the TNN layer constitute majority of the hardware complexity. Table 2 provides the hardware complexity estimates for three TNN designs (average, largest and smallest) based on the characteristic equations and technology scaling from [6]. Our area and power metrics scale linearly with the synapse count while the latency scales logarithmically. Thus, the dimensionality reduction via random projections reduces the area and power by 77.5% and the latency by up to 30%. The average synapse count for the proposed TNN architectures across the 36 datasets is 970, which translates to about 5,000 μm^2 die area, 22 μW power and 5 ns processing delay in 7 nm CMOS. The largest TNN network (*WordSynonyms*) with 6,750 synapses incurs 33,000 μm^2 and 155 μW. On the other hand, the smallest TNN network (*SonyAIRobotSurface2*) with 130 synapses consumes less than 1,000 μm^2 and 2 μW.

5. CONCLUSION

We propose a TNN-based approach for online, unsupervised clustering of real-time sensory signals that is ideal for always-on, edge-native devices. The proposed architecture is capable of iterative continuous learning and can easily be scaled for multivariate signals. Our method outperforms K-means, and is competitive with state-of-the-art algorithms when tested across a wide variety of time-series datasets. Furthermore, sizable TNNs can be directly implemented in 7 nm CMOS with less than 1 mm^2 die area and 1 mW power. To the best of our knowledge, this is the first work that applies TNNs to the problem of univariate time-series clustering. This can serve as a foundation for future neuromorphic works in industrial edge applications such as anomaly detection and keyword spotting.

6. REFERENCES

[1] Qianli Ma, Jiawei Zheng, Sen Li, and Gary W Cottrell, “Learning representations for time series clustering,” in *Advances in Neural Information Processing Systems*, 2019, pp. 3781–3791.

[2] Qin Zhang, Jia Wu, Peng Zhang, Guodong Long, and Chengqi Zhang, “Salient subsequence learning for time series clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2193–2207, 2018.

[3] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme, “Learning time-series shapelets,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 392–401.

[4] James Smith, “Space-time algebra: A model for neocortical computation,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 289–300.

[5] James E Smith, “Space-time computing with temporal neural networks,” *Synthesis Lectures on Computer Architecture*, 2017.

[6] Harideep Nair, John Paul Shen, and James E Smith, “Direct CMOS implementation of neuromorphic temporal neural networks for sensory processing,” *arXiv preprint arXiv:2009.00457*, 2020.

[7] Peter U Diehl and Matthew Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, pp. 99, 2015.

[8] Amirhossein Tavanaei and Anthony S Maida, “Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning,” *arXiv preprint arXiv:1611.03000*, 2016.

[9] Amirhossein Tavanaei and Anthony S Maida, “Multi-layer unsupervised learning in a spiking convolutional neural network,” in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2023–2030.

[10] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier, “STDP-based spiking deep convolutional neural networks for object recognition,” *Neural Networks*, vol. 99, pp. 56–67, 2018.

[11] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida, “Deep learning in spiking neural networks,” *Neural Networks*, vol. 111, pp. 47–63, 2019.

[12] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothée Masquelier, “Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks,” *Pattern Recognition*, vol. 94, pp. 87–95, 2019.

[13] Zihan Pan, Jibin Wu, Malu Zhang, Haizhou Li, and Yansong Chua, “Neural population coding for effective temporal classification,” in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[14] William B Johnson and Joram Lindenstrauss, “Extensions of Lipschitz mappings into a Hilbert space,” *Contemporary mathematics*, vol. 26, no. 189-206, pp. 1, 1984.

[15] Dimitris Achlioptas, “Database-friendly random projections: Johnson-Lindenstrauss with binary coins,” *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 671–687, 2003.

[16] Sander M Bohte, Joost N Kok, and Han La Poutre, “Error-backpropagation in temporally encoded networks of spiking neurons,” *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[17] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML, “The UCR time series classification archive,” October 2018, `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.