Implementation of Recurrent Networks Utilizing TNNs

Team 3
Ilayda Onur, Justin Nguyen
ECE Department
Carnegie Mellon University
U.S.A

***Abstract*—In this project, we design a recurrent TNN column and propose an associated STDP-based learning rule[1] for it. As our proposed design relies solely on TNNs, we have demonstrated the ability for our network to carry the benefits of both RNNs and TNNs. Some of the desirable properties such as sequential operation, memory, sparse activation, and local learning have positive impacts on the performance, energy efficiency, and applications of our network. This is especially useful in conventional machine learning applications where the data is sequential or RNNs are typically deployed. Instead of relying on RNNs which operate on dense matrices, our recurrent TNN column attempts to mimic the RNN architecture and its benefits using biologically-inspired neuromorphic design principles. We apply our recurrent TNN column as a feature extraction step in an entirely TNN based vehicle classification dataset. We are able to achieve performance near to the state-of-the-art GRU networks on this dataset.**

I. Introduction

Many of today's data-driven processing occurs on time-series data such as video, audio, text, and sensor measurements. There is a wide body of research in optimizing computation models to best interact and analyze this data. Some approaches to analyzing this type of data relies on heavy pre-processing and feature engineering while other approaches allow direct processing of the raw time-series data. Quite often, time-series data is transformed into a variety of representations to either make specific aspects of the data more expressive or be more compatible with the model which will learn the data. Some transformations include Fourier Transforms, Wavelets, Dictionary Learning, Cepstrums, and a wide array of windowing functions. While these techniques can be extremely useful in adapting data to the machine learning model, these strategies often trade temporal resolution for better resolution in another basis. This is usually done to better fit the model or allow the model to learn the data more easily.

Rather than transforming the data to fit the model, there exists models which match the sequential nature of time-series data. One type of model is Recurrent Neural Networks (RNNs) which is a class of Artificial Neural Network (ANN) which explicitly models these temporal sequences. [8] RNNs have the ability to consume sequences incrementally unlike other ANN architectures (feed-forward, convolutional, etc.) which rely on fixed sized inputs. RNNs have been extremely successful in several fields such as NLP, object detection, financial market analysis, and temporal anomaly detection. [10] This is especially useful in low-latency or low-memory applications. Additionally, RNNs are easily composable atop of other ANN architectures which allows machine learning applications to incorporate memory into their model.

It is clear that there exist many benefits to using RNNs, especially when it comes to time-series data processing. With the recent growth of distributed sensor networks (IOT), smart cities, and integration of intelligent systems into our every day tasks, the push for machine learning on the edge is booming. This recent focus is the result of consumers wanting faster and more energy efficient applications in their everyday lives; computing on the edge achieves this by reducing latency, minimizing communication, and maximizing system power efficiency. [13] [15] The aspect of neuromorphic computing as a biologically inspired computing paradigm meets all of the above requirements for intelligent processing on the edge: sparse processing has immense power savings, local learning allows for flexible and more intelligent applications, and neuromorphic systems can run on modern hardware at very high speeds. [14] The ability to natively process time-series data can be an invaluable tool for neuromorphic computing. Given the power and local learning advantages available in neuromorphic designs, the ability to interface directly with time-series data is desired; potentially some of the added benefits of RNNs could be translated to Temporal Neural Networks (TNNs) as well such as reduced network sizes, complexity, or fan-out.

In this work, we propose a recurrent TNN column where the recurrent connections are learned jointly with the feed-forward connections using a modified version of the STDP (spike time dependent plasticity) learning rules already prevalent in the field of neuromorphic computing. We then apply this model to a low-dimensional time-series classification task and evaluate the performance of our network against a state of the art GRU (Gated Recurrent Unit) RNN model. [4]

II. Related Works

There are three specific fields relating to implementation of RNNs using TNNs. The first is a work by Diehl et al. which implements a pre-trained RNN cell in neuromorphic hardware. [6] While feasible for modern deployments as a potential drop-in replacement of hardware which can implement

[1]The source code for our project can be found here: https://github.com/justinnuwin/18-847_Project

RNNs, this approach still relies on dense inputs and ”global” back-propagation based training of the traditional RNN. Our work instead focuses on incorporating a recurrent architecture into the TNN architecture and training it with biologically feasible learning rules common in the neuromorphic space such as STDP.

Reservoir computing is another field which is highly related to implementing recurrent architecture in TNNs. Reservoir computing is commonly implemented using Spiking Neural Networks (SNNs) where the ”reservoir” contains an extremely large collection of neurons and random connections between those neurons (and connections forming loops are allowed). The reservoir is not necessarily trained, but the provides a transformation of input data into an extremely high-dimensional input space via the random connections. This technique is inspired by how biological brains function and has been shown to be successful. [17] [7] This technique is heavily dependent on the initialization of the reservoir and since the reservoir is untrained, the task of classification is pushed to the decoder (usually a classical statistical classifier) entirely. Additionally the size of the reservoir may require a high level of complexity in hardware implementation.

The final area of research that is related to our work is in applications utilizing SNNs or TNNs and utilize recurrent connections to carry temporal information in a network which, alone, does not carry any notion of time. A work by Xing et al. proposes a Spiking Convolutional Recurrent Neural Network which takes the well studied Spiking Convolutional Neural Network and augments it with recurrent connections to enable it to have a sense of memory. [18] As discussed previously, the RNN architecture is extremely flexible in being able to be composed atop of other models, which has proved RNNs to be an invaluable tool in both traditional machine learning and neuromorphic systems. Our work varies from Xing in that we will be using the RNN as the primary model for feature processing and managing the temporal aspect of the data.

III. Methodology

To attempt implementing this recurrent architecture using TNNs, we took the straight forward approach and mimicked concepts from traditional RNNs using TNN columns. This in-of its self is not novel, but a large focus of our work will be on how the recurrent synapses can be trained with STDP, and how the features that are generated from this recurrent TNN column are meaningful and can be classified.

We will first discuss the dataset, base line, and metrics that our network was applied on. Then we detail our design process chronologically and the challenges at each stage of our network's design until we arrive at the final architecture. At each step we will analyze the performance of our network against the baseline using the outlined metrics.

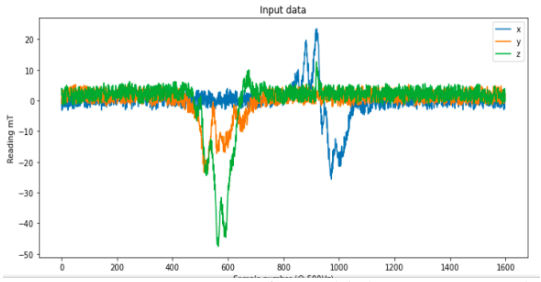

Fig. 1: Magnetometer Trace for vehicle #7 (Jeep Cherokee SUV) driving in speed class #1 (20 MPH).

A. Dataset

The dataset used in this work is a vehicle classification dataset[2], selected for our previous experience with this dataset and a working RNN (GRU and LSTM) implementation. The dataset consists of measurements taken from a pavement mounted 3-axis magnetometer sampling disturbances in the local magnetic field at a sampling rate of 500Hz. When a vehicle drives besides the sensor, the metal within the vehicle induce a disturbance in the magnetic field which the magnetometer measures. These magnetometer traces are characterized by a near zero steady state level (since the bias from the Earth's magnetic field removed during preprocessing), except when the car passes by which generates a short-term (0.5s - 0.15s depending on the vehicle and vehicle speed) high-amplitude transient. There are 8 distinct vehicles (1 police cruiser, 5 sedans, 2 SUVs) and the measurements are taken in 5 speed classes (20 - 80 MPH). An example magnetometer trace is shown in Figure 1. The magnetometer used to collect this data has a measurement range of $\pm$1200uT (though the dataset presents values scaled to $\pm$120 for internal processing) and internal 16-bit ADC.

Throughout this work we evaluated our network against this dataset in three ways. The first method was a simplified version of the dataset containing only the two most distinctive classes. These two classes were vehicles 1 (a Chevy police cruiser which is an extremely heavy vehicle) and 4 (Kia Optima sedan). The second method was another simplified version of the dataset containing only the two most similar classes. The classes for this method were vehicles 4 and 2 (Chevy sedan). The final variation of the dataset the network was trained on was the entire dataset.

B. Baseline and Metrics

As mentioned in Section III-A, this particular dataset was selected since we had access and experience with a working RNN model for this data which was designed to be deployed on an embedded microcontroller platform. Our work, focusing on neuromorphic implementation of the RNN architecture using TNNs, is targeted towards embedded applications which can leverage all the advantages of neuromorphic computing

[2]This dataset was collected by Dr. Bob Iannucci at CMU Silicon Valley. Please reach out to him for access to this dataset: bob@sv.cmu.edu

paradigm outlined in Section I. The baseline network is a small GRU network with a softmax fully-connected layer for classification; it is implemented using Tensorflow Lite for Microcontrollers (TFLM). The baseline consists of less than 1000 tunable parameters, and this network when trained on the raw dataset can achieve 99% accuracy. This baseline does not prove to be a reliable comparison for our network which utilizes a recurrent TNN since the TNNs operate on discrete values while ANNs utilize floating point values.[3] An experiment was run to provide a more equal comparison between the two network implementations; the GRU model was trained on a downsampled version of the dataset (with a decimation factor of 64) resulting with 96% accuracy and the GRU model was also trained on a quantized version of the data, binning to the nearest 15 and 10 (selection of these values are discussed in Section IV-D), which resulted in accuracies of 80% and 89% respectively. The comparison using quantized data provides a more fair and true assessment of the neuromorphic implementation since the quantization process inherently requires loss.

The GRU network was deployed onto an ATMEL SAMD51 which is a 32-bit Cortex M4 microcontroller with integrated floating point unit and operates at 120MHz. The network is able to classify a vehicle in ≈0.2 seconds and consumes ≈54mW.

To compare the two implementations of RNNs, accuracy, sensitivity, ease of implementation/training, and estimated real world performance will be used.

IV. Network Architecture 1

A. Recurrent Architecture

Our initial 2-layer-architecture is displayed in Figure 2. The first layer is an unsupervised STDP layer whereas the second layer is a supervised RSTDP layer. [16] [11] [1] Input data is encoded using one of the encoding schemes that is outlined below in IV-C and IV-D. The three encoded channels are concatenated together with the previous hidden state to form the input neurons. The hidden and output layers are trained separately and both have Winner-Takes-All inhibition. [16]

For this architecture, the best network performance is achieved by bin encoding scheme outlined in IV-D and we kept the bin resolution as 16. A total of 32 (16 for positive amplitudes, 16 for negative amplitudes) neurons represented each encoded channel data and created a concatenated encoded input data size of 96. This network performed ≈ 52% accuracy with input neuron size 111, hidden size 15, output size 8 and threshold set to 0.9. Parameters used for STDP are explained in Section IV-E and Table I.

B. Challenges and Observations

1) Masking: We observed two masking effects as follows. Firstly, the largest amplitude encountered in the sequence "masked" any smaller feature that followed. In other words, the weights saturated into the highest amplitude even after additional relatively high peaks were seen. Secondly, the channel that contained the highest amplitude masked the other two channels, inhibiting the model to learn features from these two channels.

2) Limit on unique patterns: As we increased the size of the hidden layer, the new neurons did not fire and contribute to the learning. When the weights were plotted for these new neurons, we realized that patterns were repeated and extra neurons did not learn new features. We reasoned it was due to the masking effect which is described above.

3) History vs Current input: The value of the firing threshold decides whether history or the current input has more effect. With a low threshold, if recurrent neurons spike earlier then history has more effect, else the current input data has more effect. However, with high threshold, the current input always has more impact. We decided to have a low threshold and weigh the history more than the current input data in order to keep the model's memory longer.

4) Short memory: The model has very short memory and forgets high peaks if they occur earlier in the sequence.

5) Time delays: Introducing time delays to the recurrent connections cause the model to learn differently. We experimented with no delay, a constant time delay of one and random delays. With random delays, the model failed to keep a notion of memory and with a constant time delay of one, the model had a very short memory. The best performance was achieved with no time delays, therefore, we kept a zero-time-delay for the rest of our work.

C. Positive-Negative Encoding

Our first approach to interfacing the continuous data to the TNN column was a positive-encoding scheme. [1] In this encoding strategy, each channel is represented by 2 nodes, a positive and negative node. The amplitude at each time-step is quantized to match the time resolution of the network. The sign of the amplitude decides which of the pair of neurons will spike for each channel. With this encoding scheme, the spike time codes the quantized amplitude.

In our experiments, we found this approach to be not effective at allowing the TNN to learn useful features in the data for several reasons. Firstly, the features generated by this scheme is one-hot for each channel which is extremely sparse. Secondly, the quantization process loses too much spatial information. Lastly, this encoding scheme does not directly encode the temporal aspect of the data. Each set of spikes at every time-step is independent of adjacent time-steps.

D. Bin-Encoding

The approach that we used to interface the data to our TNN is bin-encoding. [5] In this encoding strategy, each channel is represented by n nodes where each node corresponds to a range in the input space. At each time-step, the amplitude is "binned" into the corresponding range, and the spike time at the firing neuron corresponds to the location within the range.

[3]The baseline model does utilize integer quantization, but this is a simple remapping of the input space into a quantized integer space by minimizing reconstruction error. This process utilizes a representative dataset and the fully trained model to build the transformation.

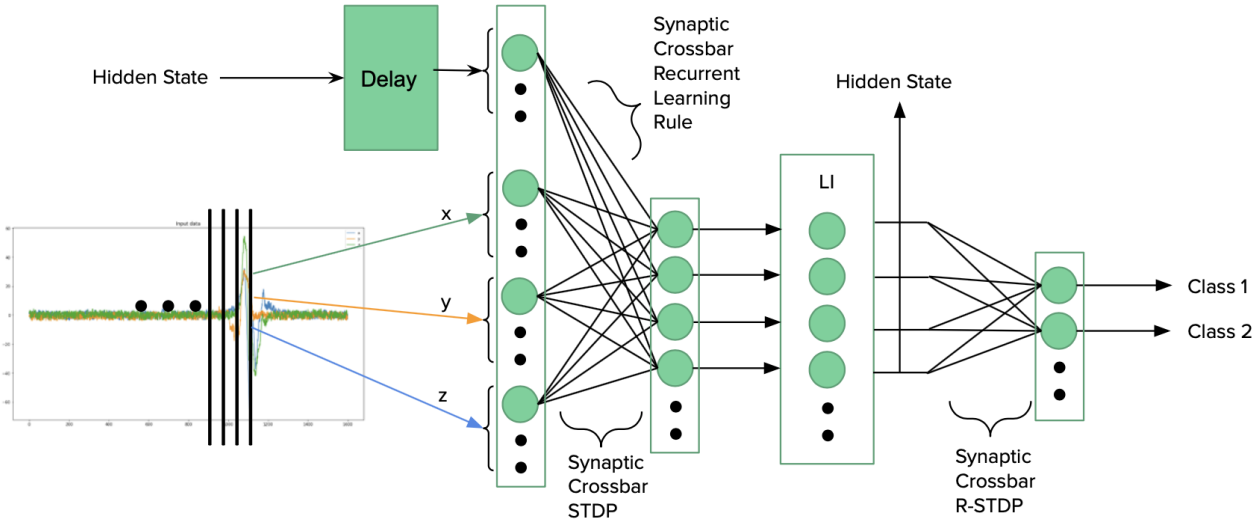

Fig. 2: Network Architecture 1

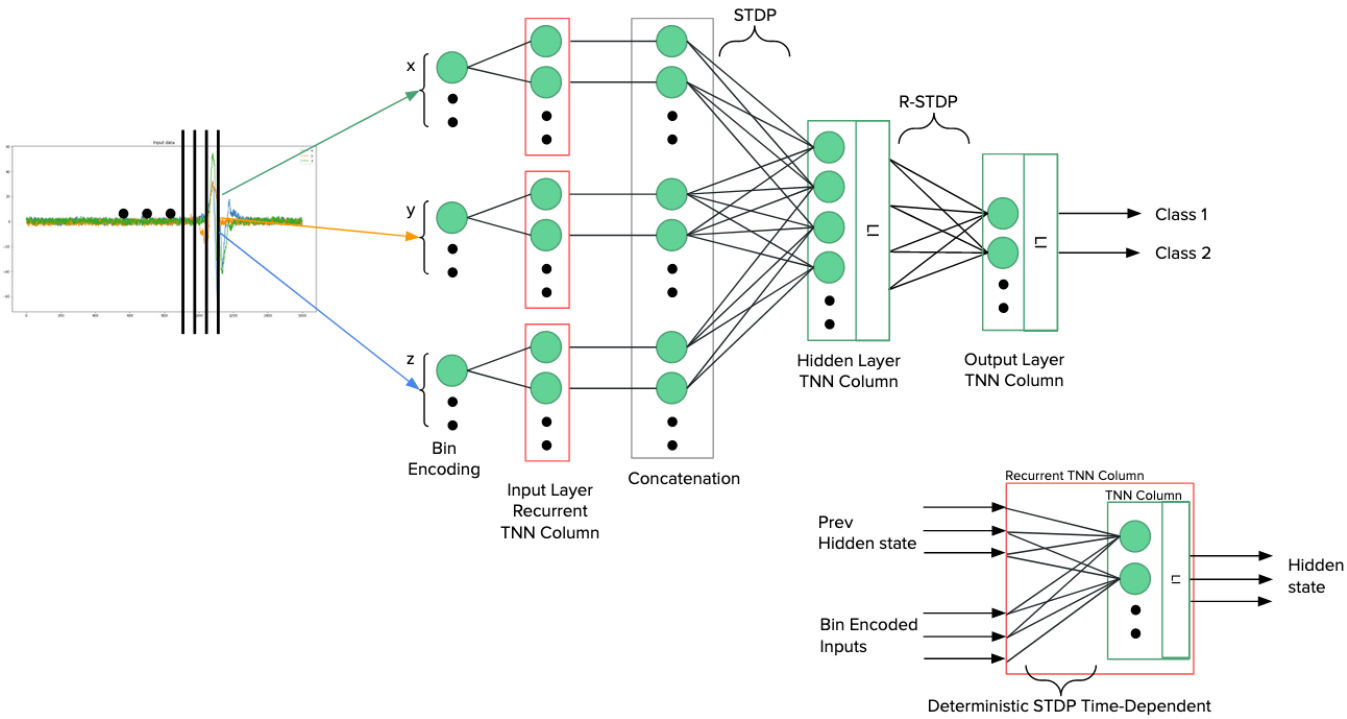

Fig. 3: Network Architecture 2

This encoding scheme is could also be compared to work by Shreyas et al., but instead of gaussian windows, our work utilizes rectangular windows without overlap instead. [12] This has several advantages. Firstly the quantization occurs at a much smaller scale. Secondly, compared to encoding schemes of other works, this approach can still be applied at individual time-steps without use of any temporal windowing functions.

This encoding scheme yielded positive results as the network now has sufficient data to learn meaningful features, but still does not solve the encoding of the temporal aspect of the data. This last problem is discussed in further detail in Section VII-B

E. Modified STDP

We experimented with various versions of the STDP learning rule and decided on the one which performed the best for

Input Conditions		Input Data Weight Update	Recurrent Weight Update	Description
$x_i \neq \infty$;	$x_i \leq z_j$	$\Delta w_{ij} = f_+(z_j - x_i\ ,\ u_{maxcapture}\ ,\ u_{mincapture})$	$\Delta w_{ij} = f_+(z_j - x_i\ ,\ u_{maxcapture}\ ,\ u_{mincapture})$	capture
$z_j \neq \infty$	$x_i > z_j$	$\Delta w_{ij} = f_-(x_i - z_j\ ,\ u_{maxminus}\ ,\ u_{minminus})$	$\Delta w_{ij} = f_-(x_i - z_j\ ,\ u_{maxminus}\ ,\ u_{minminus})$	minus
$x_i \neq \infty\ ;\ z_j = \infty$		$\Delta w_{ij} = -u_{ibackoff}$	$\Delta w_{ij} = -u_{rbackoff}$	backoff
$x_i = \infty\ ;\ z_j \neq \infty$		$\Delta w_{ij} = u_{search}$	$\Delta w_{ij} = 0$	search
$x_i = \infty\ ;\ z_j = \infty$		$\Delta w_{ij} = 0$	$\Delta w_{ij} = 0$	do nothing

TABLE I: STDP learning rule used for the model based off of the learning rules proposed by Nair et al. [11]. Update functions for capture and minus are as follows:

$$f_+(\Delta t, u_{max}, u_{min}) = (\Delta t + 1) * [\frac{u_{max} - u_{min}}{t_{max}}] \tag{3}$$

$$f_-(\Delta t, u_{max}, u_{min}) = -(\Delta t + 1) * [\frac{u_{max} - u_{min}}{t_{max}}] \tag{4}$$

where t_{max} is the maximum spike time. Positive weight updates are higher for earlier spikes than later ones. Negative weight updates are higher for later spikes than earlier ones. Parameters used in the model are $u_{maxcapture}$ = 0.007, $u_{mincapture}$ = 0.004, $u_{maxminus}$ = 0.021, $u_{minminus}$ = 0.019, $u_{ibackoff}$ = 0.00001, $u_{rbackoff}$ = 0.008 and u_{search} = 0.00005

this time-series data. The STDP rule that we used for the rest of our work is described in Table I.

Our STDP learning rule is slightly different for the input data connections than for the recurrent connections and weight updates are based on the time difference between the pre and post synaptic spikes. Firstly, we have a very small backoff learning rate for the input data connections, $u_{ibackoff} \ll u_{rbackoff}$, as time-series input comes in one-by-one. In addition, we do not have a concept of search for the recurrent connections.

We initialize the weight matrix randomly for the recurrent connections and keep the initial weights as zero for the input data connections. Weights lie in the range [0,1].

V. Network Architecture 2

A. Separate Input Encoders

In order to avoid the masking issues outlined under the challenges of the Network Architecture 1, we split up the network into a separate column for each channel. Figure 3 shows the new 3-layer architecture. First two layers are unsupervised, the last layer is supervised learning.

There are three separate recurrent TNN columns for each channel. These three separate recurrent TNN columns form the Input Layer Recurrent TNN Column. Each recurrent TNN column has connections from its associated bin encoded channel data and previous hidden state. In the first step, individual recurrent TNN columns are trained separately via STDP with Winner-Takes-All inhibition. Once the weights converge for all three recurrent TNN columns, the second layer is trained via STDP with Winner-Takes-All inhibition. Inputs to the hidden layer TNN column are the winning neurons of each recurrent TNN column. Last step is the RSTDP learning with Winner-Takes-All inhibition for the output layer.

By switching from the joint TNN to 3 TNNs, we allowed the model to learn finer details and we observed a quick performance increase from ≈ 52% to ≈ 72% using dictionary approach outlined in Section V-B. Figure 4 displays the new features the updated model learns.

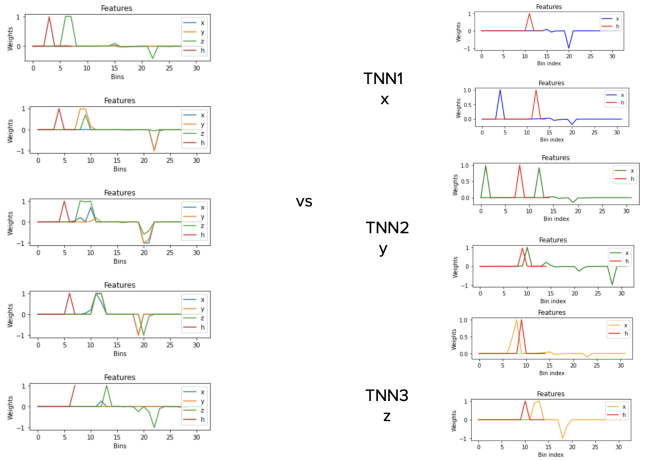

Fig. 4: Features from the joint TNN vs separate 3 TNNS. The x label corresponds to bin indices of the encoded input data. Indices 0 - 15 corresponds to positive amplitudes with 0 being the highest and 15 lowest. Indices 16 - 32 corresponds to negative amplitudes with 32 being the highest and 16 lowest.

B. Dictionary Approach

Once the encoders were trained, we wanted to evaluate how discriminative the extracted features from the 3 TNNs were. The dictionary (aka the matching matrix) counts the number of occurrences of each generated hidden state combination given the label on the training set. It forms the hidden layer TNN column of Figure 3 by acting like a mapper. However, the hidden state combinations are not preset, the model learns these pattern combinations in an unsupervised manner. Many features were highly discriminative and applying this dictionary to classify inputs in the test set yielded ≈ 93% accuracy. Table II shows example dictionary entries.

1) Cons: Dictionary approach is not a scalable solution. The size of the dictionary grows exponentially as the hidden state size or the number of channels increases. The upper bound for the dictionary size is

$$O(h^{c+1}) \tag{5}$$

where h = #hidden neurons and c = #channels.

With 15 hidden neurons and 3 channels upper bound becomes $O(15^4) \approx 50,625$

2) Pros: Dictionary approach provided us initial insights about how explanatory the models' features were. It also achieved a high performance of $\approx 93\%$ and encouraged us that our second updated architecture was good at finding relative peaks.

The actual dictionary learned from the training set turned out to be small with size $256 \approx O(15^2)$. This indicated that the patterns were discriminate of the vehicle types.

TNN			Labels							
1	2	3	0	1	2	3	4	5	6	7
Winning Neuron Index			Counts							
0	0	7	0	0	0	81	0	0	0	0
0	0	9	16	0	0	0	0	0	0	0
5	9	9	0	0	0	0	0	8	6	0
4	12	9	0	0	0	0	0	0	1	0

TABLE II: 4 examples of dictionary entries from the trained dictionary of size 256. Left side shows the winning neuron indices for each of the 3 TNNs and the right side shows the number of occurrences for each label. i) Pattern 0,0,7 is discriminative of label 3. ii) Pattern 0,0,9 is discriminative of label 0. iii) Pattern 0,0,0 is not discriminative. iv) Even the pattern 4,12,9 is seen only once, it has the power to classify the test input correctly. This is an example that dictionary learns finer details.

However, we still need to keep the upper bound for online learning as online learning needs to be robust to novel data.

C. RSTDP Voting

We experimented with two approaches in order to reduce the high hidden layer size the dictionary creates. RSTDP voting is the first approach we tried and we analyzed different versions of this method. The RSTDP voting skips the hidden layer TNN column of Figure 3 and directly connects to the output neurons. The winning neurons of each recurrent TNN column gets a reward (weight increase) for their connections to the correct label (output neuron) of the training example. Once the training is completed, each neuron's weight in the recurrent TNN columns is normalized to get a probability distribution that represents how explanatory those neurons are for each output neuron. At the inference step, predicted label is computed to be the output neuron that has the highest joint probability as shown in Equation 6.

$$\hat{y} = argmax \quad W_{h1} * W_{h2} * W_{h3} \tag{6}$$

where W_{hi} corresponds to the normalized weights of the winning neuron of the i^{th} TNN and $W_{hi} \in \mathcal{R}^8$.

We experimented with uniform and non-uniform rewards where in non-uniform rewarding rule, the neuron to spike first gets a higher voting power. Both gave us similar results, and we used uniform voting rewards of 1 for the rest of our work.

RSTDP voting reduced the complexity to

$$O(h * c) \tag{7}$$

where h = #hidden neurons and c = #channels. However, the performance was also lower with $\approx 82\%$. This is because this approach generalizes better but loses finer details that the dictionary captures. Since losing finer details led to a lower performance, we concluded that features of the recurrent TNNs are not perfectly representing the data.

D. TNN Classifier

The second design we tried in order to reduce the high upper bound from the dictionary approach is a TNN based classifier at the network output. Here, we replaced the dictionary entirely with a two feed-forward TNN columns which classify the features generated from the recurrent TNNs. With this design, the recurrent TNN columns can be viewed as the encoder side of the network.

The rationale behind two columns for classification can be explained by analyzing the dictionary of unique spike patterns generate by the recurrent TNN columns. We observe that the neurons in the recurrent column may spike for several different labels. Additionally, for some labels, a neuron spikes less often compared to others. This overlap and wide range of spike occurrences for each label means that the low-dimensional representation from the recurrent TNN columns would lose information when classified directly due to the nature of how the TNN excitatory column and STDP operate—these components tend to get saturated by common occurrences of patterns and prefer sparse activations. Due to this an approach with a single classifier layer would result in many of the fine details, dictionary entries with low volume of activations, to be lost, hence the hidden column is implemented which first transforms the features from the recurrent TNN column into a higher dimensional space via unsupervised clustering learned by stochastic STDP outlined in [12]. The classification is performed by the final TNN column which has its synapses trained using reward-based STDP (RSTDP) described in.

We train the two TNN columns in the classifier separately. When trained on the two class example containing the most discriminative vehicles, the classifier was able to achieve 88% accuracy. For the full dataset with eight classes, due to time constraints, the network was only able to achieve 52.2% accuracy, but this could likely be improved with additional training.

VI. Results

We analyzed the performance of our model for three types of test sets: all 8 classes, most similar and most dissimilar pairs. We first looked at pairs of classes to have initial insights on our model's performance. The most similar pair is label 2 (Chevy sedan) and 4 (Kia sedan) and the most dissimilar pair is label 1 (police cruiser) and 4 (Kia sedan).

Similar to Section IV-A, bin-encoding with resolution of 16 is used. For all the designs, each separate channel TNN

column has recurrent hidden layer of size 20 and use the STDP parameters outlined in Table I. For the End-to-End v2 Network design, second hidden layer size is 256.

Model results in Table IV are consistent with the complexity of the test sets. All three designs perform the best for differentiating between the most different vehicles, with the dictionary approach providing a high 0.99% accuracy. As discussed in Section V-C, lower performance of RSTDP voting against dictionary is due to loss of finer details. However, both dictionary and RSTDP perform onpar with quantized version baseline accuracy of 89%. Although the End-to-End v2 Network performs well for the 2-class problem, there is more work needed to improve the network, some approaches will be discussed in Section VII

As outlined in Table III, a neuromorphic arhitecture consumes less power and time than an artificial RNN network. This emphasizes the potential benefits of neuromorphic arhitectures in replacing artificial RNNs.

	Area (mm^2)	Power (mW)	Time/Delay (ns)
RSTDP Voting	0.017	35.96	2972
End-to-End v2 Network	0.1	36.35	2974
Baseline RNN	-	54	2e+8

TABLE III: Area, power and time/delay performance of different designs. The equations used for these calculations are taken from the paper by Nair et al. [11]. The End-to-End v2 Network has a total of 20,528 synapses and RSTDP Voting has 3,600 synapses. We included the number of time steps required to process each example, which is the sequence length of 500.

VII. Future Work

A. Ensemble Methods

From the work discussed thus far, it is clear that the network is capable of extracting discriminative features from the time-series data. At the same time, more work could still be put into the classification stage of the network. As opposed to the single classifier we have experimented with in this project, another approach could be taken where the classifier network is an ensemble of classifiers. In this strategy, the ensemble combines the results of multiple weak-classifiers to provide better accuracy. As seen in the analysis on our dictionary approach in Section V-B, it is clear that even the intermediate features from the network are still selective. Each classifier need not be restricted to TNN architecture nor multi-class classification either. Classical classifiers could be utilized. Additionally, multi-class classification architectures by combining multiple binary classifiers trained in a one-vs-all or one-vs-one manner.

B. Better Encoding scheme

While discussing how bin-encoding tries to solve the challenges of Positive-Negative Encoding in Section IV-D, one problem that bin-encoding still does not address is the encoding of temporal data. At face value, the network is still implementing memory and performing recurrence in its incremental operation, but the representation of temporal aspect of the data could still be vastly improved. When first transitioning from Positive-Negative Encoding to bin-encoding, the primary problem being solved was interfacing the spatial resolution to the network in a reliable manner. This was done by minimizing the amount of data loss when the results were quantized. With this encoding scheme, the absolute spike time representing the position within the range does not explicitly encode any useful data since the specific position within the range does not mean much when the data is already quantized into the n bins when using bin-encoding. The spike time in TNNs is used to represent the response time to a certain stimuli – where the lower response time (lower latency) represents a stronger or more important feature. The use of position in the range does not fit this paradigm, and may give similar or worse performance compared to using constant or random spike times as any particular position in the bin has no intrinsic importance compared to another position within the bin.

One alternative encoding scheme is using 3-hot codes for the firing neurons with constant or random spike times. As seen in this work, the network still achieves notable results with the bin position spike times. In this project, we did not analyze the impact this aspect of the encoding scheme played into the network performance, but with the current analysis of the encoding scheme, it is quite clear that it would have similar results to constant or random spike timings. Since the rationale behind selecting bin-encoding was improving spatial resolution and feature discriminability, a 3-hot encoding could be used. This encoding scheme adds to the current bin-encoding scheme with the addition of the adjacent bins to the firing neuron activate too. Experimenting with variations of this approach with constant spike timings, or center-weighted spike timings (adjacent bins spike with larger spike times) could provide better performance as the signal is better localized and explicitly encoded as a position within the larger overall range.

Another alternative encoding scheme which explicitly models the temporal sequence and best utilizes the importance or response-strength paradigm inherent to absolute spike times is having the spike time proportional to the derivative of the signal at the current time-step. With this approach, the current firing index still represents the amplitude of the signal at the current time step at the same quantized resolution discussed throughout this work, but with the added benefit of incorporating information from the previous time-step and estimate of the future time step in the spike time. Using the response-strength view of the spike time, it can be argued that faster moving signals (a larger instantaneous derivative) could be represented as a stronger response (smaller spike-time or latency) compared to slower signals.

C. Additional Datasets

In order to analyze the robustness of our model, we need to observe how our model performs for additional datasets where amplitude is a distinctive feature. We have couple of datasets

		8 Classes	Labels 1 & 4	Labels 2 & 4
Dictionary	Purity	0.93169	0.9957	0.976
	Coverage	1.0	1.0.	1.0.
	Accuracy	0.93169	0.9957	0.976
RSTDP Voting	Purity	0.8253	0.884	0.861
	Coverage	1.0	1.0	1.0
	Accuracy	0.8253	0.884	0.861
End-to-End v2 Network	Purity	0.5439	0.8801	0.8237
	Coverage	1.0	1.0	1.0
	Accuracy	0.5439	0.8801	0.8237

TABLE IV: Results for different designs

that we think can be used to test our model.

The first dataset is Human Activity Recognition Using Smartphones Data Set. [2] This dataset contains six activities such as walking and standing up and the data consists of measurements taken from embedded accelerometer and gyroscope. However, the challenge with this dataset is that the data is a single stream that contains different activity types. Identifying transitions between different activities is a difficult task on its own.

The second relevant dataset is Occupancy Detection Dataset. [3]. This is a binary classification problem for detecting whether the room is occupied or not. Dataset contains 6 features such as light intensity and humidity and from the published papers it is concluded that relative amplitudes play a key role.

D. Reservoir Computing

As discussed in Section II, reservoir computing is a mathematical framework that is becoming an important research topic. Reservoir computing is a type of recurrent neural network that has a reservoir of spiking neurons that map the input data to a high dimension. Although a couple of papers [9] use STDP to let reservoir neurons self-organize, most papers initialize the reservoir connections randomly and leave the reservoir weights untrained. Work is mostly done for solving speech recognition problems and to our knowledge, there is no end-to-end reservoir model which uses a temporal classifier (STDP or RSTDP) as its decoder. In addition, to our knowledge, all the reservoir computing models have rate encoded spiking neurons and there is no work based on time encoded reservoir spiking neurons. A future work is to experiment with a time-encoded reservoir architecture for our dataset.

VIII. Conclusions

Over the duration of this project, we were able to design a recurrent TNN column and associated learning rule. This allowed the network to extract distinctive features from the time-series dataset, which are classifiable by a separate TNN network with performance near to the state-of-the-art GRU networks on this dataset. As our proposed design relies solely on TNNs, we have demonstrated the ability for our network to carry the benefits of both RNNs and TNNs. Some of the desirable properties such as sequential operation, memory, sparse activation, and local learning have positive impacts on the performance, energy efficiency, and applications of our network. Specifically in this project our recurrent TNN architecture has been shown to be able to learn relative peaks and amplitudes and consume data incrementally without the need for windowing functions. Some challenges still remain in this area of research such as finding better encoding schemes that mesh well with the dataset and finding better methodologies for training these types of networks as STDP other Hebbian-learning based rules may not always give optimal results.

To our knowledge, no prior work has been done on a purely temporal recurrent network architecture that has time encoded spiking neurons. In this aspect, our work presents a novel model and our results are promising. Artificial RNNs achieve high accuracies for time-series data because these networks have the ability to capture information from floating numbers without losing resolution. Through this project, we had the opportunity to analyze how discretizing the space as well as time domain performs for time-series data. Our results achieved to convey that TNNs can exploit the temporal features of time-series data.

Acknowledgements

We would like to thank Shreyas Chaudhari and Harideep Nair for their advice and direction throughout this project. We would also like to thank Dr. John Shen and Dr. James E. Smith for teaching us about neuromorphic computing and helpful critiques of our project. Finally we would like to thank Dr. Iannucci for giving us access to the dataset and trained RNN models to experiment with.

References

[1] A. Amirshahi and M. Hashemi, "Ecg classification algorithm based on stdp and r-stdp neural networks for real-time monitoring on ultra low-power personal wearable devices," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1483–1493, 2019.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *ESANN*, 2013.

[3] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO 2 measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28–39, jan 2016. [Online]. Available: https://doi.org/10.1016%2Fj.enbuild.2015.11.071

[4] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[5] G. Dellaferrera, F. Martinelli, and M. Cernak, "A bin encoding training of a spiking neural network-based voice activity detection," 2019.

[6] P. U. Diehl, G. Zarrella, A. S. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," *CoRR*, vol. abs/1601.04187, 2016. [Online]. Available: http://arxiv.org/abs/1601.04187

[7] A. Gilra and W. Gerstner, "Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network," *eLife*, vol. 6, Nov. 2017. [Online]. Available: https://doi.org/10.7554/elife.28295

[8] M. Hüsken and P. Stagge, "Recurrent neural networks for time series classification," *Neurocomputing*, vol. 50, pp. 223–235, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231201007068

[9] Y. Jin and P. Li, "Ap-stdp: A novel self-organizing mechanism for efficient reservoir computing," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 1158–1165.

[10] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[11] H. Nair, J. P. Shen, and J. E. Smith, "Direct CMOS implementation of neuromorphic temporal neural networks for sensory processing," *CoRR*, vol. abs/2009.00457, 2020. [Online]. Available: https://arxiv.org/abs/2009.00457

[12] J. M. J. S. S. Chaudhari, H. Nair, "Unsupervised clustering of time series signals using neuromorphic energy-efficient temporal neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, 2021.

[13] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[14] J. P. Shen, "Lecture on neuromorphic computer architecture: Background motivation ("why this course now?")," February 2021.

[15] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[16] J. E. Smith, "A temporal neural network architecture for online learning," *CoRR*, vol. abs/2011.13844, 2020. [Online]. Available: https://arxiv.org/abs/2011.13844

[17] N. Soures and D. Kudithipudi, "Spiking reservoir networks: Brain-inspired recurrent algorithms that use random, fixed synaptic strengths," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 78–87, 2019.

[18] Y. Xing, G. Di Caterina, and J. Soraghan, "A new spiking convolutional recurrent neural network (scrnn) with applications to event-based hand gesture recognition," *Frontiers in Neuroscience*, vol. 14, p. 1143, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.590164