Real-time classification of LIDAR data using discrete-time Recurrent Spiking Neural Networks

Anca-Diana Vicol*
CWI
Amsterdam, The Netherlands
ancadiana.research@gmail.com

Bojian Yin
CWI
Amsterdam, The Netherlands
Bojian.Yin@cwi.nl

Sander M. Bohté
CWI
Amsterdam, The Netherlands
S.M.Bohte@cwi.nl

***Abstract*—With the advancement of Edge AI and autonomous systems, AI applications are increasingly subject to energy, latency and environmental constraints. Biological neural systems naturally adhere to these constraints and, as such are a source of inspiration. Spiking Neural Networks (SNNs) are a more detailed model of biological neural processing. Recent work shows that they perform well in object recognition and detection in general, and in Autonomous Driving tasks based on ranged LIDAR data in particular. However, these LIDAR-SNN approaches do not optimize for latency, as they require the entire frame to be scanned before processing. They also require large SNNs, limiting the energy efficiency achieved. To reach both low-latency and high energy efficiency in LIDAR object recognition, we develop a compact recurrent SNN. First, we propose and examine an open LIDAR labeled dataset by processing the point clouds from the KITTI Vision Benchmark. We then train our recurrent SNNs on this dataset and propose specific optimizations, including input encoding, sparse connectivity and truncation of error-backpropagation. With these optimizations, we show that compact recurrent SNNs can exceed the performance of classical RNNs like LSTMs and approach the performance of large non-spiking CNNs. Additionally, they significantly reduce latency by allowing early and online object classification before the end of the sequence.**

***Index Terms*—Spiking Neural Networks, LIDAR, Object Recognition**

I. Introduction

With the increasing attention for untethered AI applications – *Edge AI* – there are multiple factors to consider when developing such solutions, from the energy usage of autonomous systems, to the latency of time-sensitive decisions, to the interference of environmental conditions on the sensors. For the technology to scale well, these requirements have to be integrated with the design from the ground up, and may require the use of specialized tools such as event-based sensors. LIDAR, in particular, can provide exact distance measurements and is less affected by lighting conditions, making it a key sensor for the Autonomous Driving industry. However, equipping Autonomous Systems with such instruments needs efficient algorithms that can process and make decisions on LIDAR data with low latency and high energy efficiency.

Latency, in particular, can benefit: considering the sensor used for the KITTI dataset as an example, it can rotate at up to 20Hz, with a significant loss in resolution (down to 1042 points per revolution). To get its full resolution (4167 points per revolution), the sensor needs to spin at 5Hz. Classifying objects online, throughout the scanning process, would potentially significantly increase the reaction speed of a vehicle while allowing the sensors to move at a slower pace and provide sharper input signals.

Spiking Neural Networks (SNN) are a variant of Artificial Neural Networks based on more detailed models of biological neurons. Neurons in SNNs communicate through binary signals – spikes – and can encode information in the relative timings of these spikes. The asynchronous, event-based and sparse communication of SNNs can lead to sizably reduced energy requirements when running these networks on specialized neuromorphic hardware [1].

To achieve both low latency and high energy efficiency, we turn to a recurrent variant of Spiking Neural Networks that is highly suitable for processing time series and that can directly process raw LIDAR data as input. We show that a discrete-time Recurrent Spiking Neural Network (RSNN) can efficiently classify LIDAR data, in real-time, throughout the scanning process. This setup would assist an Autonomous System in confidently perceiving the environment before a full revolution of the sensor is finished. Moreover, predicting object types before they are fully scanned could allow a system to make low latency decisions.

We develop a reproducible and challenging benchmark for online LIDAR processing from the KITTI Vision Benchmark, by extracting point clouds representing objects and creating a LIDAR object recognition dataset. On this labeled dataset we train a discrete-time RSNN and study performance, latency and energy efficiency.

To achieve satisfactory classification performance, we study the effect of different input encoding methods on the performance of the network. We further show effective solutions can be optimized for better accuracy, shorter training times and better energy efficiency. In particular, we show that large sparsely connected SNNs achieve better accuracy than smaller networks with the same number of non-zero parameters. We present the benefits of using Truncated Backpropagation-Through-Time [2], which improves the training time of the networks, and can have a positive effect on the accuracy.

* Now at Google.

A. Background

SNNs are comprised of spiking neurons that capture varying degrees of complex processing in biological neurons. The Leaky-Integrate-and-Fire (LIF) spiking neuron [3, Chapter 4] is a simple phenomenological spiking neuron model that is used in many SNNs. Its behaviour is described by a single variable equation (1) and associated spiking condition:

$$\begin{aligned} \tau_m \frac{du}{dt} &= -u(t) + RI(t), \\ S(t) &= \hat{f}_s\left(u(t), \upsilon\right) \\ u(t) &= u(t)\left(1 - S(t)\right) + u_{reset} S(t), \end{aligned} \tag{1}$$

where u represents the membrane potential of the neuron, τ_m is the neuron's time constant, such that $\tau_m = RC$, and υ is the threshold the membrane potential has to reach for an output spike to be created; $S(t)$ is the binary spike indicator, u_{reset} denotes the potential to which the emission of a spike resets the membrane potential, and $\hat{f}()$ is the spike-function. Compared to other spiking neuron models, the LIF preserves only a few features of biological spiking neurons [4], but has low implementation cost.

Training SNNs is particularly challenging due to the discontinuous nature of the spiking mechanism, which makes such networks not amenable to standard gradient descent and error-backpropagation as generally used for training Artificial Neural Networks [5]. Various solutions to learning in SNNs have been proposed, including switching to another learning style [6], using a different method of encoding information within the network [7], or changing the way gradients are computed [8], [9]. Most alternatives for computing the gradient tend to be intimately tied to how information is represented by spikes. For example, one can compute the exact time of each spike and then use these continuous values for gradient computation, or consider discrete time steps in the network and approximate the gradient for any given step. The former method has been successfully used in multiple projects [5], [8], [10], including related work on LIDAR data [11]. However, by using such temporal encoding, these networks are generally constrained to one spike per cell for every input, which is not desirable in a setting where information is continuously processed.

The gradient approximation method, also used to train our networks, inserts a function for the non-existing gradient through the spike-function, as a "Surrogate Gradient" [9] [12]. Many choices for this function have been examined in the literature, such as a piece-wise linear function, the derivative of a sigmoid or the exponential function [9]. The Surrogate Gradient method provides flexibility in implementation, in particular in relation to existing ANN frameworks like PyTorch and Tensorflow, and enables convenient handling of both spatial and temporal credit assignment.

B. Related work

LIDAR scans the distance to objects in the environment by sweeping the surrounding area with a stream of laser pulses and measuring the return time to work out the distance for each pulse and angle. A full scan is done progressively for different azimuths and elevations.

Previous solutions for solving LIDAR based object-recognition tasks have required the data to be fully scanned and then processed into a point cloud that can then be the input of a classification system. Early work by Himmelsbach et al. [13] split the frame into a grid, computed the connected components of the cells and then used the results for object segmentation and feature computation, which became the input of an SVM classifier. In SqueezeSeg [14], the whole point cloud is projected to a 2D image and then fed into a CNN, which outputs point-wise labels to detect three types of objects: cars, pedestrians and cyclists. Prokhorov [15] processes the point cloud as a time series of points, which represents the input to a small Recurrent Neural Network that classifies objects as vehicles and non-vehicles. This approach can be used through the scanning process. All these ANN-based solutions are not analyzed from the point of view of energy consumption, which is a strong advantage of SNNs as well as an important factor in autonomous systems.

Zhou et al. and Wang et al. [11] [16] [17] propose SNN-based solutions for object detection and classification from LIDAR data. In [11], image classification is carried out using a modest-sized feed-forward network applied to a simple synthetic dataset obtained by simulating a LIDAR sensor that perceives different types of roads populated with cars, pedestrians and trucks. In [16] the authors expand this approach using feed-forward and convolutional SNNs for object recognition in three datasets: simulated LIDAR, objects extracted from the KITTI dataset and DVS_barrel [18]. Furthermore, [17] contains a Spiking Convolutional Neural Network approach to object detection, by incorporating the YOLOv2 architecture. This is tested on the bird's eye view and 3D detection benchmarks from the KITTI dataset.

Both [17] and [16] contain promising estimates for the energy consumption the networks would have if they were running on neuromorphic hardware. All of these SNNs networks have the constraint that each neuron is only allowed to spike once, which complicates effective implementations of recurrent connections and processing incoming information as a time series.

II. Model

To sequentially process information like raw LIDAR data, we turn to recurrent SNNs as described by Yin et al. [19]: the network consists of LIF neurons simulated at discrete time points. These SNNs contain layer-wise recurrent connections and are trained using Backpropagation-Through-Time (BPTT) and Surrogate Gradients.

The neuronal behaviour of LIF neurons can be simulated at discrete time points using the forward-Euler first-order exponential integrator method, as described by (2)–(3):

$$u_t = \alpha u_{t-1} + (1-\alpha) R_m I_t - S_{t-1}\theta \tag{2}$$

$$\alpha = \exp(-dt/\tau_m), \tag{3}$$

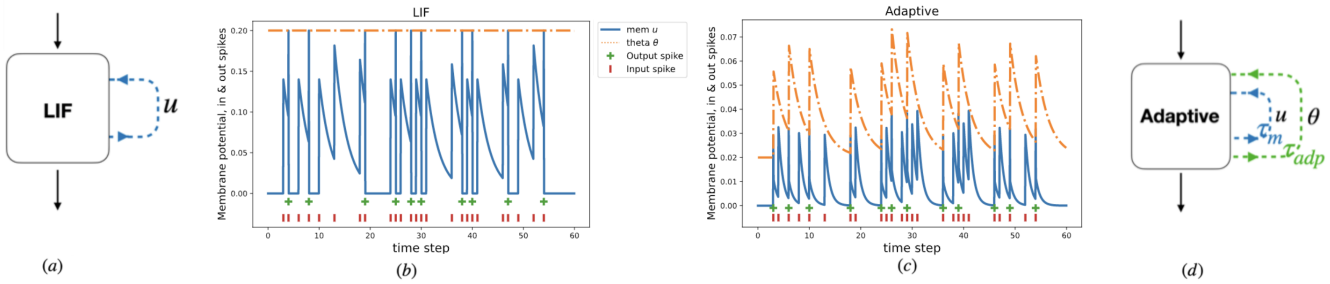


Fig. 1: LIF and Adaptive spiking neuron behaviour.(a,d) The decay of the membrane potential and adaptation can be modeled as self-recurrent connections. (b,c) Impinging input spikes (red) increase (or decrease) the membrane potential (blue) which then decays back to the resting potential (0). When the potential reaches the fixed threshold θ_0 (yellow dotted line), an output spike is emitted (green cross) and the potential is reset to the resting potential. In the Adaptive spiking neuron (c), the threshold is increased for every emitted spike, and then decays back to the resting threshold b_0.

where at each time step t, u_t represents the membrane potential of the cell, I_t the input current and S_t the binary activity of the neuron. The parameter α relates to the decay of the membrane potential, while R_m is the membrane resistance, τ_m the decay time constant of the membrane potential and θ is the threshold.

Neural adaptation is a feature of the biological neuron that can be added to the LIF model of (2). Its implementation is expressed by (4)–(6):

$$\theta = b_0 + \beta\eta_t \tag{4}$$

$$\eta_t = \rho\eta_{t-1} + (1-\rho)S_{t-1} \tag{5}$$

$$\rho = \exp(-dt/\tau_{adp}). \tag{6}$$

Here, the threshold becomes a function of time, with a fixed minimal value of b_0 and an adaptive contribution $\{\beta\eta_t\}$. The adaptive contribution decays with parameter ρ, which also depends on an additional time constant τ_{adp}. The behaviour of the variables in the LIF neurons with and without adaptation given a modulated spike-rate input is shown in Fig. 1.

The LIF and Adaptive LIF neurons are stateful cells that can be building blocks for feed-forward and recurrent networks. By replacing the gradient computation of the neuronal activation with a Surrogate Gradient, such networks can be trained using Backpropagation-Through-Time in the same manner as more traditional ANNs [9]. Our SNNs, in particular, use a difference of Gaussian functions as a Surrogate Gradient for the neuron activation, where the highest values reside around the point where the membrane potential is equal to the threshold [20]. The network is implemented using the PyTorch framework [21] and relies on the autograd feature for all gradient computations.

III. Methods

A. Dataset

The KITTI Vision Benchmark [22] was created by collecting data on static and dynamic objects in diverse scenarios, such as city streets, rural areas and freeways. The data was captured using a vehicle equipped with multiple sensors;

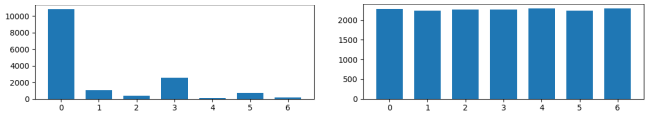


Fig. 2: Example distribution by class of the original and resampled dataset.

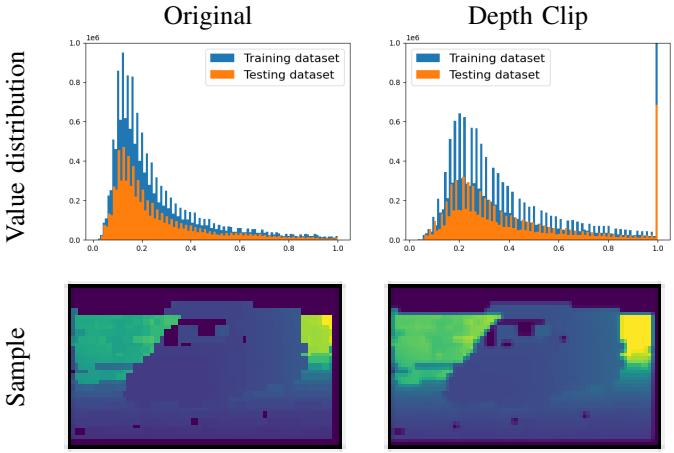


Fig. 3: Value distribution of the original and clipped data, followed by example images.

the laser scanner used for developing this benchmark is the commercially available Velodyne HDL-64E. The data from the various sensors was synchronized and processed in order to form a set of benchmarks [23], where we are using the 3D object detection benchmark.

We have developed an object extraction pipeline that creates 2D images of the same size, each containing one object, where the values of the pixels are derived from the depth of the 3D points[1]. For this, the 3D point cloud is projected to 2D and all the points that fall inside the specified window are selected.

[1]https://github.com/ancadiana23/RSNN_LIDAR

Each window is of constant size and contains the bounding box annotation of the respective object in its center. Annotated objects that are too big or too small in comparison with the window size are ignored. Then all the 3D points whose 2D projections fall inside the computed window are associated with the respective object. Thus, each object is encoded by a 2D image where the value of each pixel is associated with the depth of the corresponding point. The pixels where no points are projected are assigned the value 0, and in the cases where multiple points in the 3D point cloud project to the same pixel, the highest value is used (as in practice this gave most contrast and worked best). The images are then scaled down by a predefined factor to have the resulting objects of a particular size, and the depth values in the dataset are normalized to be between 0.0 and 1.0.

Analyzing the class and value distribution of the resulting dataset shows that the dataset class distribution is severely imbalanced (Fig. 2). To resolve this, we use the torch WeightedRandomSampler in order to create a balanced training set. Testing uses the test set with the original class distribution in most experiments, unless explicitly stated that the test set is also resampled.

We also find that the resulting pixel values are mostly clustered in the interval [0.1, 0.4], with a long tail using the remaining dynamic range [0.4, 1.0]. To increase the effectively used dynamic range, we apply a depth clip to all the values, where we empirically select the maximum depth from the original values. These depth values are then normalized to be between 0.0 and 1.0. The result of this process can be observed in Fig. 3, which presents the value distributions and sample images from the original and depth clipped set.

The dataset generation configuration was empirically determined by visually inspecting the results. The size of the windows applied to the 2D image is 232x392, which are then resized by a factor of 8, resulting in samples of size 30x50. We ignore any object that is either larger than the window or smaller than a fifth of the window. Since the testing examples of the KITTI benchmark are not publicly available, we used the first 5000 frames for constructing a training dataset and the last 2481 examples as our test dataset. As there can be multiple objects in the same frame, this procedure results in having 15892 extracted objects for training and 7926 objects in our testing set.

B. Depth encoding

When classifying the objects in the dataset, we use a sliding window whose stride and size can be adjusted through model hyper-parameters. For each point the sensor measures the delay between emitting and receiving a laser beam, from which the distance to the reflecting surface can be computed. Thus, the input to an object classification model can be either a binary signal at the receiving time, or the value of the time delay as a floating point number.

In order to simulate these two input types, we use the temporal and rate encoding respectively. Rate encoding signifies that the network receives the values described in the dataset section directly as current injection into the input neurons.

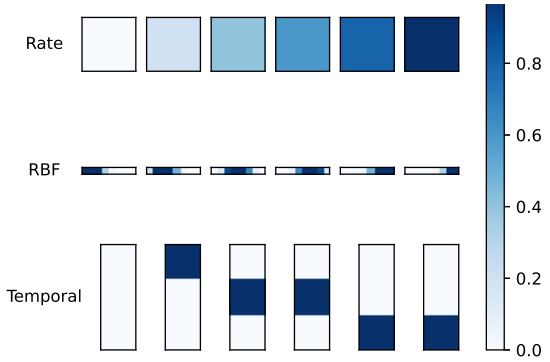


Fig. 4: Encoding methods. The first row shows 6 equally spaced values between 0.0 and 1.0, which are examples of the raw values. The following rows show the output of different encoding methods, where the horizontal space represents neighbouring cells in one layer, and the vertical space signifies different time steps.

Our temporal encoding method receives float values which are translated into binary spikes, at specific discrete times, thus relaying one input window over multiple time steps. The value interval (0.0, 1.0] is split into equal segments, and then each value is replaced by a binary signal at a time step corresponding to the value's segment (Fig. 4). The values strictly equal to 0.0 correspond to inactive input cells.

In addition to the two input types, rate and temporal encoding, we also use Radial Basis Functions (RBF) [24]. This type of Receptive Field Coding is a population coding method that uses multiple cells with different receptive fields to encode each input value. The receptive fields used here are described by Gaussian functions of identical shapes but with different means. Thus, for small input values, the leftmost neurons are activated, while larger values move the activation towards the right of the neuron population (Fig. 4).

Experimentally, we find that the effectiveness of RBF encodings varies depending on the number of neurons used to represent each pixel. Smaller values, such as 4 neurons per pixel, show less over-fitting but also decreased performance. Higher values however result in lower training accuracy and much lower testing accuracy. For the experiments with RBF encoding, we selected 8 neurons per pixel as the optimal value.

TABLE I: Weight sparsity comparison. For each sparsity level here are the total number of parameters (weights), the number of non-zero weights, and the number of neurons. The default network here contains 128 and 64 cells in the hidden layers.

Sparsity Level	#Param	#Non-Zero Param	#Cell
0 %	35520	35520	199
30 %	48564	34033	235
50 %	66831	33343	278
70 %	107235	32082	356
80 %	158444	31584	436
90 %	307242	30884	613

C. Sparsity

Recent work has shown that making Recurrent Neural Networks sparser can be advantageous not only for expediting the training process but also for improving performance [25]. Distinctively, in [26], [27], the authors show that keeping a constant number of non-zero parameters while increasing the size and sparsity of a network leads to increased accuracy. They do so by creating larger networks at the beginning of the training process and populating a binary mask throughout the training process, always setting the smallest weights to zero. Here, we test this idea in the context of RSNNs, by randomly constructing a binary mask at the beginning of the training process. Starting with the default sized networks (two layer RSNNs, see section IV), the layers are then scaled and masked with a probability p such that they use an approximately constant number of non-zero parameters. The mask remains constant throughout the training process. The sparsity and corresponding network sizes are shown in Table I.

D. Truncated Backpropagation-Through-Time

Even with our down-sampled dataset, event sequences are relatively long, creating issues in training and testing the model with BPTT; these issues would only be exacerbated by scaling to a real-life object detection task. To mitigate this, we implement Truncated Backpropagation-Through-Time (T-BPTT). T-BPTT splits a long sequence into several segments and then computes the gradient and weight updates only using one segment at a time. This training optimization is selected when training the model and the number of backpropagation steps is controlled by a hyper-parameter, *backprop_step* .

In detail, BPTT follows the flow of information through time in reverse, unfolding the recurrent influences in the network, including self-recurrences in the neural cell: Fig. 5 shows the connections of one cell unfolded in time. The membrane potential and adaptive threshold create self-recurrent connections for each cell, which are depicted in pink and yellow, respectively. There is also a layer-wise recurrent connection (green) from the spikes in the previous time step to the membrane potential of the current one. The inset in Fig. 5 zooms in on the actual computation, and shows the interaction between variables from (2) – (6): the red parameters are learned, and the recurrent connections are represented in the same colors in all three views of the network.

To apply truncation of the unfolded sequence, we detach the internal parameters of the model from the computational graph every *backprop_step* number of time steps, such that future gradients with respect to these variables will not be backpropagated past the specified time point. Importantly, all variables have to be detached, including the membrane potentials and the firing activities of the cells, as they also persist through multiple time steps. Moreover, these variables effectively construct the memory of the network, and thus should maintain their values and not be reinitialized at truncation.

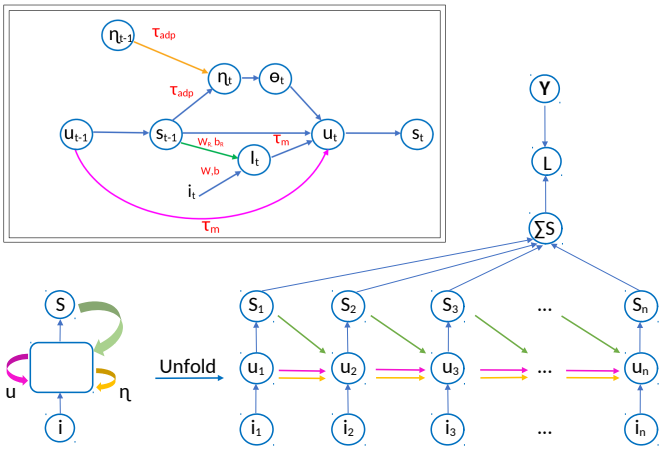


Fig. 5: Recurrent SNN with adaptive LIF neurons. The membrane potential (pink) and the adaptive threshold (yellow) are represented as self recurrent connections. The layer-wise recurrent connections are represented by the green arrows. Inset: detailed graph for spiking neuron variables and parameters.

IV. Experiments and Results

To test the encoding and optimization methods, as well as the efficiency of the RSNN approach, we construct a default RSNN containing two hidden layers of 128 and 64 cells, respectively, and two traditional ANNs for comparison: an RNN with two LSTM layers of 128 neurons and one fully-connected layer (architecture optimized for the task); the second is a CNN with 2 convolutional layers and 3 fully connected ones. For all the Recurrent Networks (RNN and SNN) we split the images into time sequences using a sliding window, which by default covers one row at a time (window = stride = 50 px).

TABLE II: Accuracy and number of active neurons for the best performing models (mean results from 5 trained networks). All RSNN networks are of default parameter size.

Model	Test Accuracy	Spikes/Frame
Rate	80.4	1127
Rate + sparsity(90%)	79.3	3355
RBF	83.7	1235
RBF + sparsity(90%)	**88.2**	3458
Temporal Coding (4)	75.7	2944
LSTM	84.0%	N/A
CNN	92.3%	N/A

A. Overall Performance

The RSNN and RNN models and their accuracy on the original test set are presented in Table II. The RSNN results surpass those of the LSTM and approach that of the frame-based CNN: pairing RBF encoding and 90% sparsity results in 88.2% test accuracy.

At the same network size in terms of parameters, rate-coding and temporal coding are substantially less accurate.

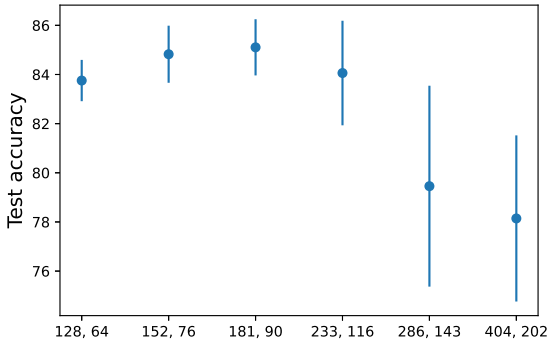


Fig. 6: Non-sparse RBF network accuracy as a function of network size.

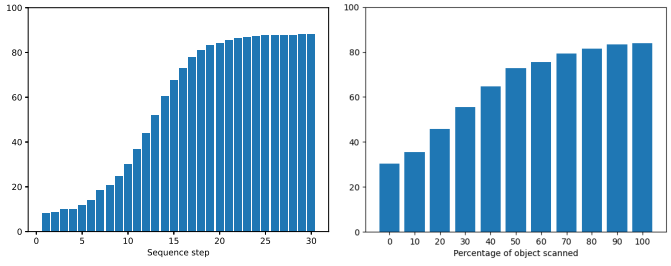


Fig. 7: Network performance over sequence steps (left) and over the object scan percentage (right).

At the same time however, rate-coding in particular requires substantially fewer spikes per frame for classification.

We also find that for non-sparse RBF-RNNs, modest sized networks are needed, with most network sizes performing approximately as well as the default-size network (Fig 6). Overall, these results confirm that RSNNs are able to successfully classify LIDAR data.

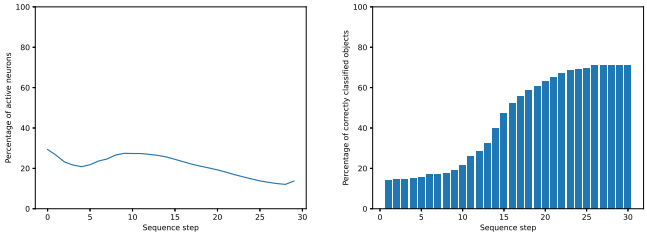


Fig. 8: Network activity (left) and performance (right) as a function of sequence steps.

B. Early classification

The sequential nature of processing in RSNNs enables the network to classify objects already during the scanning process and provide tentative results before the end of the input sequence. Fig. 7 shows the classification performance along the sequence, where the model's accuracy is plotted at different sequence steps or scanning percentages. The left plot presents a monotonically increasing accuracy over the length of the sequence, demonstrating that the RSNN is able to classify partial inputs correctly. Viewed as a function of the object, the right plot shows the accuracy of the network against the percentages of object area scanned. For this plot, we use the original 2D bounding boxes and plot percentages scanned in order to normalize over objects of multiple sizes – we note again the monotonous increase in accuracy. An example of the resulting online classification is shown in Fig. 9.

We furthermore study the average activity in the network while processing a sequence. Fig. 8 presents on the left the mean percentage of active neurons over the sequence, and on the right the model's accuracy over the sequence. We find that on average around 20% of neurons emit a spike at any time step. There is also a noticeable increase in activity between time steps 5 and 15, which corresponds to a steep increase in accuracy.

These findings suggest that, in a practical setting, not only would the RSNN model be able to classify objects before the end of a 360° scan, but it would also be capable of predicting its type before the object is fully scanned.

C. Sparsity

As shown in Table II, we find that (when using the RBF encoding) network performance is improved for increased network sparsity. This technique maintains a fixed number of network non-zero parameters - and increases the number of neural cells. At the same time, we find that networks with sparse connections show increasing average network activity, especially for very sparsely connected networks (Fig. 11).

To further explore its effects, we plot the mean and standard deviation of the network's accuracy for network size and sparsity, as calculated over the last 25 epochs of training and test accuracy. Fig. 10 shows that sparsity improves the network accuracy when using the RBF encoding, but it is not beneficial with rate-coded input. Our experiments revealed that sparsity works well with rate coding only on the resampled test set. We speculate that larger sparser networks manage better class separation due to their larger latent space. When having a highly skewed testing set, class separation is less important than learning to classify the majority class, and thus the network has good results without using sparsity; however, in this case, spreading the predictions over more classes can lead to worse results.

Overall, using the RBF encoding consistently results in better accuracy, especially on the skewed (original) test set, which leads us to believe that this coding method improves the class separation for the majority class. Thus, when both sparsity and RBF are employed they improve the performance of the network regardless of the testing set.

D. Truncated Backpropagation-Trough-Time

The effect of Truncated Backpropagation-Trough-Time on the training process is shown in Fig. 12, plotting the training time in seconds from experiments with various input encodings

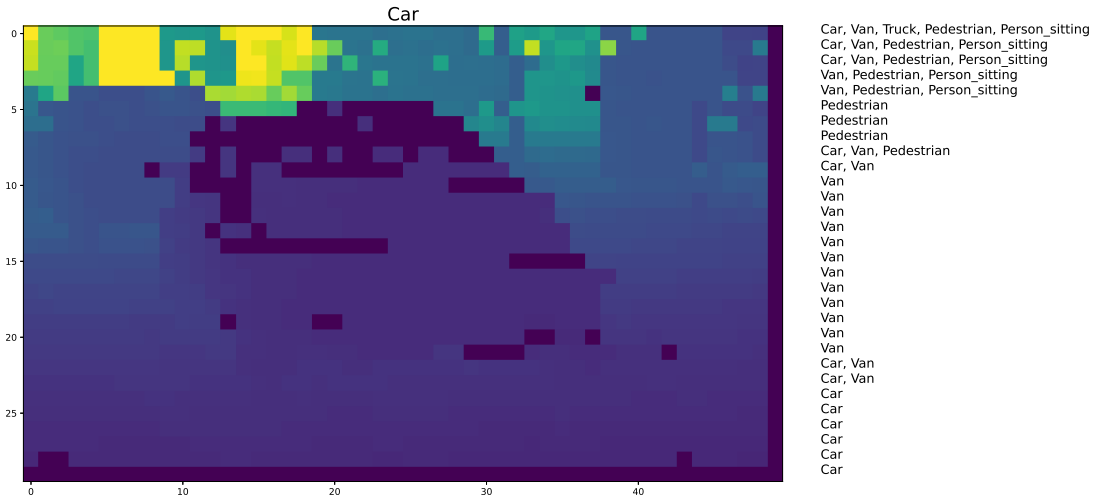
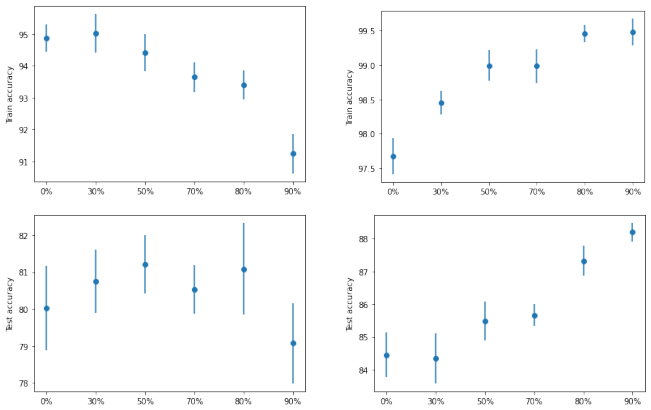


Fig. 9: Examples of network output over row by row sequences. Above the image is the correct label of the object, and to the right of each row is the network output after processing the row.



a) Rate b) RBF

Fig. 10: Sparsity effect on the original test set for rate-coding (left) and RBF(8) coding (right).

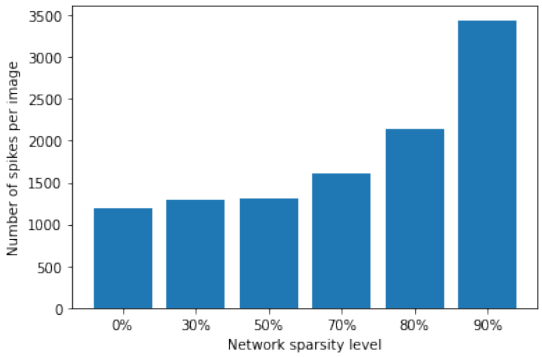


Fig. 11: Number of spikes per image for RBF models at different levels of sparsity.

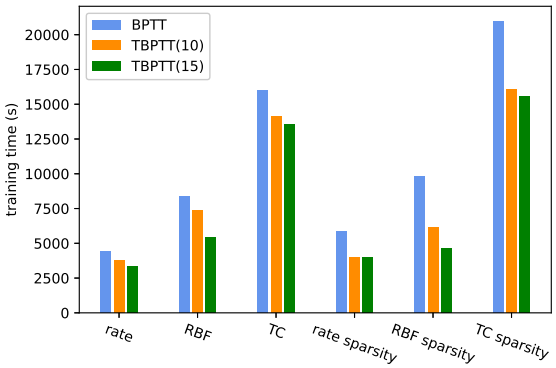


Fig. 12: Training times using BPTT and T-BPTT with 10 and 15 steps truncation intervals.

for the default RSNNs. Invariably, we find that using Truncated Backprop Through Time leads to shorter training times.

The effect of T-BPTT on testing and training set accuracies is shown in Fig. 13, when using RBF coding and varying length for the truncation intervals. We find that truncation leads to relatively good results, approaching full BPTT accuracy, with 10 steps proving to be the sweet spot for this particular encoding method.

We conclude that Truncated Backpropagation-Through-Time always shortens the training time of the model. Concerning performance, however, it can be a useful tool in certain cases.

V. Energy Requirements

Next to classification accuracy, energy requirements of the networks are an important secondary concern, as the practical applications of such systems are Autonomous System often subject to energy constraints.

We study the energy requirements of the network in more detail by approximating the number of required computations. Table III calculates the energy usage of a particular network layer with respect to the energy of multiply-and-accumulate operations (MAC) and accumulate operations (AC) [19]. With

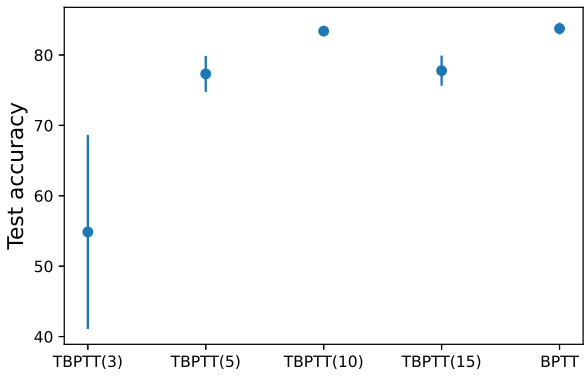


Fig. 13: The mean testing accuracies of the last 25 epochs of training, using T-BPTT with different truncation intervals and RBF(8) encoding.

TABLE III: Energy requirement formulas for different types of NN layers. Here E_{AC} and E_{MAC} represent the energy cost per AC and MAC respectively, m is the input size and n is the output size of the layer, while Fr is the firing rate.

Network	**Layer type**	**Energy/Layer**
SNN	Adaptive	$(mn + 2n)E_{AC}Fr + 2nE_{MAC}$
	Adaptive & Recurrent	$(mn + nn + 2n)E_{AC}Fr + 2nE_{MAC}$
Traditional NN	FC	mnE_{MAC}
	LSTM	$(4mn + 4nn + 3n)E_{MAC}$

these equations, we calculate the corresponding energy use for the LSTM, default RSNN, and best performing RSNN. As calculated in Table IV, we find that then that the theoretical energy required by the default RSNN is one order of magnitude lower than that of a traditional recurrent network even before factoring in the sparse activation rates. Using 20% firing rates the energy usage of both SNNs falls well below that of the RNN.

TABLE IV: Approximate amount of energy required to process one input (one line of an image) using a traditional recurrent neural network, compared to a spiking neural network. The networks used in these computations are the ones presented in the previous experiments. The theoretical approximations are computed using the computation complexity formulas from Table III, while the practical ones depend on the Intel Loihi measurements [28]

	Encoding	Theoretical	Practical
RNN	-	223,872 E_{MAC}	-
SNN 128 × 64	Rate	35,918FrE_{AC} +398E_{MAC}	11 nJ
SNN 90% sparse	RBF	449,868FrE_{AC} +1226E_{MAC}	35 nJ

We can also approximate the energy requirements the SNN would have when implemented on actual neuromorphic hardware, using the data published by the Intel Loihi Team [28]. They measured the energy of a neuronal update when the cell is active/inactive as 81 pJ and 52 pJ, respectively. Our default RSNN has 199 cells resulting in 11nJ energy usage per step when running with 20% firing rate. The larger, better performing RSNN with 90% sparsity and RBF(8) encoding reaches 35nJ usage with the same firing rate. In the scenario where a Velodyne sensor was moving at 20Hz with 50 images per 360° frame, running the larger SNN would require about 1mW. The caveat of this calculation is that a Loihi implementation of our approach has yet to be constructed. Thus, we do not have the certainty that this network's time steps would match onto the neuronal updates used in their measurements.

VI. Discussion

We find that we can train compact RSNNs to successfully processes LIDAR data using various types of input encoding, where adding an RBF-style population encoding significantly improves the network's results. We also find that network sparsity improves the RSNN performance when using a) rate encoding and training and testing sets with the same class distributions; b) RBF encoding. Even though the number of non-zero parameters stays the same, the number of cells grows with the size of the network. At the same time, the number of spikes per frame grows rapidly with the increasing number of neurons in the network, which leads to higher computational and energy requirements for such sparse networks.

Overall, we demonstrated that RSNNs successfully classifies the provided LIDAR data, reaching over 88% accuracy on the testing set, exceeding the performance of classical RNNs like LSTMs and approaching that of non-sequential CNNs which process the entire frame. We find moreover the RSNNs can preemptively classify examples before the end of the sequence and even before the target object is fully perceived.

While theoretically highly efficient, the practical approximation of energy consumption is a rough estimation, and an actual implementation on neuromorphic hardware, such as the Intel Loihi [28], would be required to confirm this estimate. Building on the developed solution, we also note that the input to the network can be expanded to also contain the reflectance values provided by LIDAR, which may improve performance further. Other multimodal solutions could also be considered, such as aggregating data from the multiple sensors equipped in an Autonomous Driving Setup.

References

[1] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers in Neuroscience*, vol. 14, 2020.

[2] C. Tallec and Y. Ollivier, "Unbiasing truncated backpropagation through time," *arXiv preprint arXiv:1705.08209*, 2017.

[3] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[4] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[5] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[6] W. Gerstner, *Biological Learning: Synaptic Plasticity, Hebb Rule and Spike TimingDependent Plasticity*, pp. 111–132. Boston, MA: Springer US, 2010.

[7] A. Grüning and S. M. Bohte, "Spiking neural networks: Principles and challenges.," in *ESANN*, Bruges, 2014.

[8] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8529–8533, IEEE, 2020.

[9] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[10] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017.

[11] S. Zhou and W. Wang, "Object detection based on lidar temporal pulses using spiking neural networks," *arXiv preprint arXiv:1810.12436*, 2018.

[12] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural Computation*, vol. 33, no. 4, pp. 899–925, 2021.

[13] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3d object perception," in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008.

[14] B. Wu, A. Wan, X. Yue, and K. Keutzer, "Squeezeseg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3d lidar point cloud," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1887–1893, IEEE, 2018.

[15] D. V. Prokhorov, "Object recognition in 3d lidar data with recurrent neural network," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 9–15, IEEE, 2009.

[16] W. Wang, S. Zhou, J. Li, X. Li, J. Yuan, and Z. Jin, "Temporal pulses driven spiking neural network for time and power efficient object recognition in autonomous driving," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 6359–6366, 2021.

[17] S. Zhou, Y. Chen, X. Li, and A. Sanyal, "Deep SCNN-based real-time object detection for self-driving vehicles using lidar temporal data," *IEEE Access*, 2020.

[18] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "Hfirst: A temporal approach to object recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 10, pp. 2028–2040, 2015.

[19] B. Yin, F. Corradi, and S. M. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, pp. 1–8, 2020.

[20] B. Yin, F. Corradi, and S. M. Bohte, "Accurate online training of dynamical spiking neural networks through forward propagation through time," *arXiv preprint arXiv:2112.11231*, 2021.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[22] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[23] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.

[24] S. M. Bohte, H. La Poutré, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," *IEEE Transactions on neural networks*, vol. 13, no. 2, pp. 426–435, 2002.

[25] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[26] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," in *International Conference on Machine Learning*, pp. 2410–2419, PMLR, 2018.

[27] J. Menick, E. Elsen, U. Evci, S. Osindero, K. Simonyan, and A. Graves, "A practical sparse approximation for real time recurrent learning," *arXiv preprint arXiv:2006.07232*, 2020.

[28] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.