UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DEVELOPMENT OF SPIKING NEURAL NETWORKS BASED ON DEEP LEARNING AND INFORMATION THEORY

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

JHON ALEJANDRO INTRIAGO CHICHANDA

PROFESORES GUÍA:
NANCY HITSCHFELD KAHLER
PABLO ESTÉVEZ VALENCIA

MIEMBROS DE LA COMISIÓN:
BENJAMIN BUSTOS CÁRDENAS
GONZALO ACUÑA LEIVA
FELIPE TOBAR HENRÍQUEZ

Este trabajo ha sido parcialmente financiado por:
Secretaría de Educación Superior, Ciencia, Tecnología e Innovación, SENESCYT
CONICYT, Fondecyt 1171678

SANTIAGO DE CHILE
2021

RESUMEN

Las *Redes Neuronales Spiking* (SNNs) son una posible vía para acortar la distancia entre el aprendizaje profundo y la neurociencia. Estos modelos se consideran redes neuronales biológicamente plausibles porque intentan simular cómo las neuronas biológicas transmiten la información. Su implementación en hardware especializado ofrece un gran potencial de velocidad computacional. Sin embargo, es necesario mejorar su rendimiento con respecto a las redes neuronales recurrentes tradicionales tales como Vanilla Recurrent Neural Network (V-RNN), Long Short Term Memory (LSTM) y Gated Recurrent Unit (GRU). Para ello, abordamos algunas debilidades de estas redes desarrollando variantes basadas en principios de aprendizaje profundo y teoría de la información. En concreto, proponemos un nuevo método heurístico para estimar los parámetros que definen al modelo SNN. También añadimos memoria externa a estos para una comparación justa con los modelos tradicionales y, finalmente, adaptamos el método *Information Bottleneck* como criterio de optimización. Estas mejoras propuestas se evaluaron con dos conjuntos de datos: El problema de los bits de paridad y el conjunto de datos Spiking Heidelberg (SHD). Nuestros principales resultados sostienen que el modelo SNN Recurrente (SRNN) mejorado logró un rendimiento similar al modelo GRU con menos parámetros en la clasificación del conjunto de datos SHD, en el que ambos modelos alcanzaron un 86% de exactitud en promedio. Además, cuando se añadió Information Bottleneck como criterio de optimización, el modelo SRNN convergió más rápido que los modelos entrenados con entropía cruzada (Cross-Entropy). Nuestros resultados sugieren que el modelo SRNN inspirado en conceptos biológicos tiene un rendimiento similar al de las RNN tradicionales cuando las condiciones de evaluación son similares (por ejemplo, utilizando el mismo número de parámetros). Además, cuando las RNNs se entrenan utilizando técnicas de control de gradientes, estos modelos mejoran su rendimiento.

ABSTRACT

Spiking Neural Networks (SNNs) are a possible way to bridge the gap between deep learning and neuroscience. These models are considered biologically plausible neural networks because they try to simulate how biological neurons transmit information. Their implementation on specialised hardware offers great potential for computational speed. However, its performance needs to be improved with respect to traditional Recurrent Neural Networks (RNNs) such as Vanilla Recurrent Neural Network (V-RNN), Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). For this purpose, we address some weaknesses of SNNs by developing variants based on deep learning and information theory principles. Specifically, we propose a new heuristic method for estimating SNN parameters. We also add external memory to SNNs for fair comparison with traditional RNNs and, finally, we adapt the Information Bottleneck method as an optimisation criterion. The proposed improvements were evaluated with two datasets: The parity bit problem and the Spiking Heidelberg dataset (SHD). Our main results show that the improved Recurrent SNN (SRNN) achieved similar performance to the GRU model with fewer parameters on the task of classifying the SHD dataset, where both models achieved 86% accuracy on average. Furthermore, when the Information Bottleneck was added, the SRNN model converged faster than the models trained with cross-entropy. Our results suggest that the SRNN model inspired by biological concepts performs similarly to traditional RNNs when the evaluation conditions are similar (e.g., using the same number of parameters). Moreover, when RNNs are trained using techniques to control gradients, these models improve their performance.

In the form of acronyms, this work goes to: LOML & FFP.

Agradecimientos

A lo largo de la redacción de esta tesis, he recibido una gran cantidad de apoyo y asistencia.

En primer lugar, quisiera agradecer a mis supervisores, el profesor Pablo Estévez y a la profesora Nancy Hitschfeld, cuya experiencia fue invaluable en la formulación de las preguntas y la metodología de la investigación. Sus comentarios, siempre oportunos, me impulsaron a mejorar mi pensamiento crítico y a realizar con éxito el presente trabajo.

También me gustaría agradecer a mis padres por sus sabios consejos y comprensión. Finalmente, no podría haber completado esta disertación sin el apoyo del LOML Cinthia, con quien siempre tuve discusiones alentadoras, así como momentos felices que me permitieron descansar la mente fuera de mi investigación.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep learning has made significant advancements in numerous applications, such as computer vision and natural language processing, applied in a variety of domains, such as medicine, biology, government, among others. It is a field of machine learning that relies on the collection of algorithms for modeling high-level abstractions in the data with multiple nonlinear transformations [15]. In view of the promising results of this leading machine learning approach, it is important to study *Artificial Neural Networks* (ANNs), as they are at the heart of it.

ANNs are series and parallel combination of processing units, capable of acquiring information through a learning process, and storing it in their connections [26]. They are inspired by biological neurons; nevertheless, most ANNs follow a model that weakly simulates the natural behaviour, mainly in the way information is propagated. These artificial networks propagate information represented in real numbers, while biological neurons propagate it using sparse spiking signals over time [31], [61].

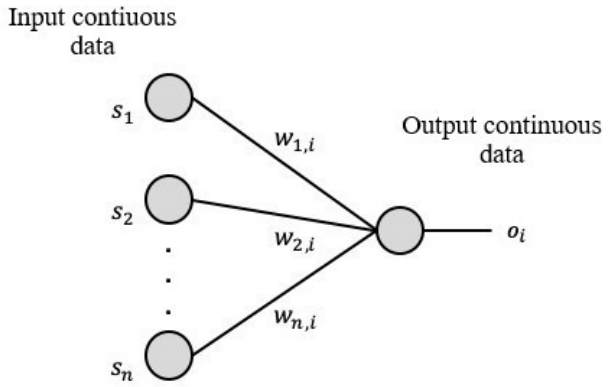
There are three components typically used in the design of the ANNs [54]. First, the *learning rule*, which drives how the synaptic connections (also known as *weights*) are updated. Second, the *optimisation criterion* or *loss function*, which quantifies the performance of the network in order to minimise its error or maximising its performance [11], [39], [54]. Third, the *architecture*, which defines how information flows (e.g. Feedforward, Convolutional, Recurrent, among others) [48].

Recurrent Neural Networks (RNNs) are commonly the first option for working with sequential data, given their outstanding results in several domains such as natural language, video, audio, among other forms of time series data [25], [34], [57]. They can be obtained by sampling delay differential equations [57]. This basic architecture has different variations, for example, some are built to perform specific computations [10], others are enhanced with new features (such as augmented memory) [24], and others are inspired by strong neuroscience principles (more biologically plausible[1]) [4], [12].

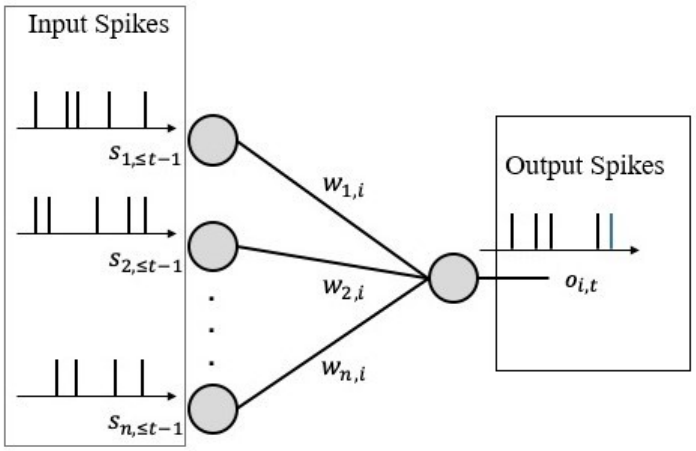
Spiking Neural Networks (SNNs) are considered biologically plausible neural networks be-

[1]When a modelled neuron is capable of producing a set of behaviours exhibited by real neurons [1]

cause they try to simulate how biological neurons transmit information (Figure 1.1). SNNs can be obtained from mathematical models such as Hodgkin and Huxley (HH) [29], Spike Response Model (SRM) [32], Leaky Integrate and Fire (LIF) [7], [20], among others. Particularly, LIF can be formally mapped to a RNN variant, known as *Spiking RNN* (SRNN), using Euler method [49]. Unlike the SNNs, the SRNNs in addition to applying a recurrence on the input data also applies a recurrence on the hidden state, i.e. it performs an explicit recurrence. SRNN is a nature versatile solution to fault-tolerant, energy-efficient signal processing [49], which makes it an object of research interest.



a) Representation of a perceptron, where S_n is the input feature, O_{i} is the output projection and $w_{n,\mathrm{i}}$ represents the learnable parameters.



b) Representation of a Spiking unit, where $S_{n,t-1}$ is the input feature at previous time step, $O_{\mathrm{i},t}$ is the output spikes projection in current time step and $w_{n,\mathrm{i}}$ represents the learnable parameters.

Figure 1.1: Illustration of a) an ANN unit and b) an SNN unit.

In general, the input data for a SRNN are binary sequences. However, they are often stored in formats that are not compatible with this model and must be transformed. These transformations are generally based on two families of neural coding: *Rate Code* [60] and *Temporal Code* [11]. Recent evidence suggests that these transformations have several replicability issues due to their parameters. For that reason, Cramer et al. [14] proposed a

benchmark with binary sequences. This benchmark belongs to a supervised learning task[2], and consists of two audio datasets mapped to spike trains through a transmitter pool based on hair cells[3].

Both SRNNs and RNNs frequently use Cross-Entropy (CE) and Mean Square Error (MSE) as optimisation criteria in supervised learning tasks [30], [49]. Other optimisation criteria are Van Rossum distance [58], Predictive Coding [54] and Information Bottleneck [6]. Information Bottleneck (IB) is based on information theory, which studies the rules that govern the transmission of messages through communication systems [47] (IB is described in Section 2.5.4). A neural network architecture can be considered as a communication system because it can work as a channel that receives an input signal and produces an output signal [21]. Therefore, it is justified to explore optimisation criteria based on information theory in these models.

Throughout this thesis we use the term *RNN* to refer only to those architectures that transmit information using real values, and the term *SRNN* for architectures that only transmit information using binary pulses or spikes (biologically plausible). Given this background, we focus on the SRNNs applied to binary sequential data classification, compared to some variants of RNNs.

1.1 Related Work

In this section, we present a brief introduction to Spiking Neural Networks focusing on their application and formulation, and their optimisation criterion based on information theoretic measures.

1.1.1 Spiking Neural Network Application and Formulation

Spiking Neural Networks have been tested in several classification tasks [14], [49], [50], [58], [61]. Previous studies have reported that SNNs are able to obtain a similar accuracy to RNNs to classify Neuromorphic datasets. These datasets generally contain records generated by sensors that mimic some human sense, e.g. the retinal saccades sensor [39], [50]. However, more recent work provides contradictory findings when these models are tested with more complex datasets, which ensure time dependence [14].

In general, SRNNs have different mathematical formulations due to their non-differentiable synaptic transfer functions[4], assumptions and discrete approximation [14], [39], [49], [58]. As a result, there are several variants of SRNNs such as those proposed by Cramer et al. [14] and Bellec et al. [4]. In this paper, we implement and explore the models formulated in Cramer et al. [14] and Yin et al. [70], we use these works as a reference because they have reported a score on one of the tasks used in this research.

The parameters of the SRNN play an important role in the performance of the model. For example, the threshold parameter ϑ, which generates the spikes in the hidden layers, is

[2]Learning from a set of labelled instances.

[3]Hair cells are the sensory receptors in the inner ear [46].

[4]On Machine learning is known as activation function.

sometimes arbitrarily defined [14], and in other works is defined by a differential equation using information from past spikes, which also needs an initial threshold value [4]. Indeed, other parameters of the SRNN are also arbitrarily defined.

1.1.2 Optimisation Criteria Based on Information Theory

Optimisation criteria based on information theory in supervised learning are those that use a logarithmic measure to minimise or maximise the predicted distribution with respect to an actual distribution to solve a task. For example, cross-entropy (CE) minimises entropy and information bottleneck (IB) maximises entropy to approximate the predicted probability to the actual probability distribution. These methods have a significant advantage over the most widely used optimisation criterion (Mean Square Error). In addition these criteria have the ability to be easily extended to non-linear systems and non-Gaussian data as they use the entire probability distribution of the data, whereas MSE uses only second order moments [52].

Cross-Entropy (CE) is the default option to be used in classification tasks because of its outstanding results. In addition, Cross-Entropy has been combined with Information Bottleneck (IB) for training ANNs in a different ways with interesting results [37], [66]. Reyes [53] modelled an algorithm, which adjusts the ANNs' weights layer by a layer based on the mutual information. Yu et al. [72] proposed a discrete IB criterion composed by the mutual information and the CE criterion. However, Information Bottleneck in Spiking Neural Networks has been explored in few works. Buesing and Maass [6] proposed a neural model based on IB with sigmoidal activation function and non-refractory voltage dynamics (an operation that prevents the neuron voltage from being reset). They only used a single neuron, thus there is an opportunity to adapt the IB criterion to more complex models.

The optimisation criteria based on information theory needs to compute entropy. Several attempts have been made to compute the entropy directly from data since it is nearly impossible to compute the probability density function for the majority of the real-world problems [55]. There are recent proposals to better estimate the entropy using different mechanisms such as the estimator by Yu et al. [71] and Kolchinsky et al. [37]. Interestingly, these two methods have nice properties such as computing the gradients of their mathematical operations.

The Yu et al. [71] method is based on Infinitely Divisible Kernels. It allows the computation of the entropy directly from data, thus facilitating new applications and uses of mutual information. This estimator can be used as a part of the feature selection [71], the Information Plane (IP) calculation [69], the optimisation criteria construction [72], among others. Kolchinsky et al. [37] method is based on a pairwise distance function between mixture components. These estimators have been applied to modelling the optimisation criterion, monitoring compression of the ANNs and other tasks [8], [36], [37].

1.2 Research Problem

In recent years, there has been an increasing interest in SNNs. These models are commonly evaluated in classification tasks [14], [49], [50], [58], [61]. Nevertheless, as far as we know, no previous study has investigated the impact of the hyperparameters related to the network

structure (e.g., number of layers and hidden units) and training strategies (e.g., gradient clipping and learning rate schedule) for different lengths of sequences. This knowledge may provide insight about the limits and practical advantages or disadvantages of SRNNs compared to traditional RNN models.

It is also important to address other aspects that influence the performance of these models such as the optimisation criteria, memory type and parameters. Regarding the optimisation criterion, it directly influences the adjusting of the weights during training of the model. Buesing and Maass [6] applied an optimisation criterion based on information theory (Information Bottleneck) to model SNNs; however, they focused on a single neuron, which limits its applicability to real-world problems.

RNNs with added external memory[5], such as Long Short-Term Memory (LSTM), have shown significant improvements in several tasks compared to the Vanilla RNN that uses internal memory[6] [45]. Indeed, LSTM (including its derivatives) deals with the problem of vanishing gradients and uses a mechanism to store the relevant information [45]. However, to our knowledge, the memory of the SRNN has not been defined yet. Thus, there is a need to study whether the external memory also improves SRNNs performance compared to not using it.

Recent research has evaluated SRNNs performance with different datasets, in which some define a fixed threshold [14], and others use an adaptive threshold (which requires an initial value for threshold) as the parameter to generate a spike [4]. However, the above-mentioned works do not provide a rationale for the definition of this threshold, which may imply an arbitrary choice.

1.3 Hypothesis

The goal of this section is to define the hypotheses to be empirically tested. The first hypothesis is related to the underlying parameters of the SRNN models; and the second is a fair comparison between a SRNN variant and a traditional RNN.

1.3.1 Parameter Model Definition on Spiking Models

The first hypothesis that will be tested in this work is **H1**, which claims that the SRNN models with the hyperparameters calculated using a heuristic method can obtain a higher accuracy than the SRNN models using literature fixed model parameters, solving the classification task used in this work. The null hypothesis is $\mathbf{H1}_{null}$, which claims that the SRNN models with the parameters calculated using a heuristic method have similar accuracy to the SRNN models using literature fixed model parameters, solving the classification task used in this work.

[5] A mechanism which uses a external element to store information.
[6] Mix all time step information in a latent variable.

1.3.2 Fair Comparison

The second hypothesis that will be tested in this work is **H2**, which states that the SRNN models using binary values in their layers have similar accuracy with respect to the traditional RNNs (such as GRU) using continuous values, solving the same classification task used in this work. Then, the null hypothesis that will prove is $\mathbf{H2}_{null}$, which states that the SRNN models using binary values in their hidden layers have lower accuracy with respect to the traditional RNNs (such as GRU) using continuous values, solving the same classification task used in this work.

1.4 Research Questions

This study addresses the following four main research questions:

- **RQ1:** What is the minimum number of parameters needed to solve the classification tasks used in this work, regardless of the RNN or SRNN architecture?
- **RQ2:** Can we automatically estimate the underlying hyperparameters of SRNNs based on their architecture?
- **RQ3:** Does the optimisation criterion space Information Bottleneck improve the performance of SRNNs compared to those trained with Cross-Entropy solving the classification tasks used in this work?

1.5 Objectives

1.5.1 General Objective

The main objective of this thesis is to develop Spiking Recurrent Neural Network models based on deep learning and information theory principles to classify binary sequential data.

1.5.2 Specific Objectives

- To propose a training and evaluation methodology for RNNs and SRNNs and then apply this methodology to the different architectures studied.
- To create a method to automatically estimate the threshold for the different architectures of SRNNs.
- To develop a new model of SRNN using an external memory approach.
- To adapt the Information Bottleneck using optimisation criterion to SRNNs.

1.6 Methodology

We propose a methodology for training and evaluating RNNs and SRNNs. We also provide an extensive experimental evaluation of different architectures of RNNs and SRNNs in the classification of the sequential XOR dataset and two spiking datasets from audio signals. This evaluation includes several techniques to improve the model performance such as: Gradient Clipping [74] and Early Stopping [41]. In addition, we vary the length of the time series

modifying the simulation time of spiking datasets. Finally, we use different numbers of layers and parameters.

In addition, we develop new models of SRNNs based on deep learning and information theory principles to classify binary sequential data. These models consider modifications in terms of optimisation criteria, memory type and parameters. For the optimisation criterion, we use as a starting point the idea proposed by Buesing and Maass [6] who adapted information bottleneck to the SNNs, and also the ideas presented by Yu et al. [72] who adapted it to ANNs. For adapting an external memory, we combine a derivative of the LSTM with the best SRNN variant obtained from the evaluation. Finally, we create a method to automatically estimate the threshold of the SRNNs based on their network architecture.

1.7 Contributions

In summary, this work presents the following main contributions:

- A methodology consisting of a set of steps to train and evaluate SRNNs and RNNs.
- A method to estimate the threshold of the SRNNs based on their architecture.
- A new SRNN model, inspired on a derivative of the LSTM.
- A new SRNN model with Information Bottleneck as the optimisation criterion.
- The source code of all proposed methods: experimental framework, heuristic method, external memory model and mutual information estimator.

1.8 Thesis Outline

In Chapter 2, we provide a background on the traditional and Spiking RNNs. Then, we formalise these models for supervised learning. Next, we explain techniques to train these models and finally we introduce Information Theory concepts and evaluation metrics.

In Chapter 3, we describe the methods and techniques used in our experiments and divide it in the following sections. First, we describe the proposed experimental framework. Second, we formalise the proposed heuristic method to compute SRNN model parameters. Third, we present the categorisation of SRNN memory and propose a variant that uses external memory. Finally, we formalise the adaptation of spatial Information Bottleneck to SRNNs.

In chapter 4, we present our baseline results using the proposed experimental framework for training and evaluating the models.

In Chapter 5, we expose the results of the proposed heuristic method for SRNNs. In addition, we compare the results obtained with the SRNN and traditional RNN models.

In Chapter 6, we present the results of the proposed Gated spiking model (G-SRNN) and compare it with the Gated Recurrent Unit (GRU).

Finally, in Chapter 7, we present the main conclusions about our work in terms of datasets, computation of model parameters, convergence steps and weights (learnable parameters).

Chapter 2

Theoretical Framework

This chapter provides an overview of the topics that are mentioned throughout the work, including an explanation of our notation. The first section refers to the Recurrent Neural Networks (RNNs) and their main architectures. The second section describes the Spiking Recurrent Neural Networks (SRNNs), along with their main architectures. The third section deals with supervised learning, including the concepts of optimisation criterion and accuracy evaluation metric. The final section concerns the training of RNNs, using the *Backpropagation Through Time* algorithm and some problems related to exploding and vanishing gradients.

Notation

The following notation is used consistently throughout this work to define the RNN and SRNN architectures. The notation in lower case italics represents a scalar (e.g, a), in lower case bold represents a vector (e.g, $\mathbf{a}$), and in upper case bold represents a matrix (e.g, $\mathbf{A}$). In general, the hidden state of SRNNs is denoted by s, however, in this work we use the notation h, as with RNNs.

- T: number of time steps in the sequence
- L: number of layers
- l: index of layer
- n: index of the time step
- d_x: dimension of the input signal
- d_h: dimension of the hidden signal
- d_y: dimension of the output signal
- $\mathbf{x}[n] = \mathbf{h}^{(0)}[n] \in \mathbb{R}^{d_x}$: input signal
- $\hat{\mathbf{y}}[n] = \mathbf{h}^{(L)}[n] \in \mathbb{R}^{d_y}$: output signal
- $\mathbf{h}^{(l)}[n] \in \mathbb{R}^{d_h}$: hidden signal on layer *lth*
- $\mathbf{W}^{(l)} \in \mathbb{R}^{\mathrm{d}_h \times \mathrm{d}_x}$: Weights on layer *lth* for the input signal
- $\mathbf{V}^{(l)} \in \mathbb{R}^{\mathrm{d}_h \times \mathrm{d}_h}$: Weights on layer *lth* for the hidden signal
- $\mathbf{U}^{(L)} \in \mathbb{R}^{\mathrm{d}_y \times \mathrm{d}_h}$: Weights on layer *lth* for the output signal
- $\odot$ Point-wise multiplication

2.1 Recurrent Neural Networks (RNNs)

RNNs are Turing complete models and belong to a family of neural networks designed for processing sequential data like text sentences, time-series, and other discrete sequences like biological sequences [2], [59]. There are different design strategies for RNNs depending on the task. For example, they can produce an output at each time step (with recurrent connections between hidden units or just from one time step to the hidden units at the next time step), or they can produce a single output after reading an entire sequence [23].

In principle, RNNs can store the representations of recent input events in form of activations with their feedback connections [28]. Different architectures have been proposed for modelling time-dependent phenomena [3]. A taxonomy was proposed based on the memory characteristics (e.g. internal memory, external memory, logic gates, attention mechanism), which classified RNN into four classes ordered by a inclusion relationship as Vanilla RNN $\subseteq$ Long Short-Term Memory $\subseteq$ Neural Stack $\subseteq$ Neural RAM [45].

2.1.1 Vanilla RNN (V-RNN)

It is the classic version of RNN that introduces memory by encoding the past information with a feedback connection, which flows from the hidden layer to itself. The hidden units induce a memory named *state memory* or *internal memory*, which is updated at each time step, where the current state only depends on one single compound event at the previous time step [45].

The Equation 2.1 describes the computation of a cell in a V-RNN, where a linear transformation is applied to the signal $\mathbf{h}^{(l-1)}[n]$ and the hidden signal $\mathbf{h}^{(l)}[n-1]$ over the weights $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$, respectively. Then, a non-linear function ϕ is applied point-wise over the sum of the transformed values. The non-linear function commonly used in this model is the *tanh function* (hyperbolic tangent), which is bounded between -1 and +1. The update equation for a cell in a V-RNN is:

$$\mathbf{h}^{(l)}[n] = \phi\left(\mathbf{W}^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right). \tag{2.1}$$

The Figure 2.1 describes the computational operations of a cell in a V-RNN and its transition steps. This graph only considers a single neuron in the first layer.

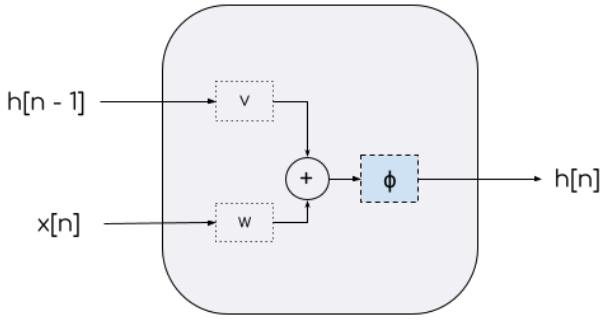


Figure 2.1: Vanilla recurrent cell.

2.1.2 Long Short-Term Memory (LSTM)

This cell was proposed to deal with the vanishing gradient problem of the V-RNN. The key insight was to incorporate non-linear, data-dependent controls into the RNN cell [57]. The innovation resides in the incorporation of external memory and gates to balance both external and internal memory. Thus, in the classical LSTM, the feedback connection of the hidden layer goes through an external memory [45].

The original LSTM prevents the gradient problems by enforcing a constant error flow through the internal states of specific neurons. Each memory cell needs two additional units to learn to open and close access to error flow. First, a multiplicative *input gate* unit, which decides whether to store certain information, protecting the memory from perturbation by irrelevant inputs. Second, a multiplicative *output gate* unit, which decides whether to access certain information, protecting other units from perturbation by currently irrelevant memory contents [28].

Then, an adaptive *forget gate* was added to the original LSTM to learn to reset memory blocks once their contents are no longer useful, thus releasing internal resources [19]. LSTM networks have different variations, but in general they use forget, input and output gates, which can be seen as a V-RNN with a sigmoid function represented by σ. The forget (Eq. 2.2) and input (Eq. 2.3) gates are used to decide the amount of information from the previous time-stamp to be used for the selection of the candidate $\mathbf{c}^{(l)}[n]$. Then, Eq. 2.6 is used to obtain the hidden signal $\mathbf{h}^{(l)}[n]$, where a point-wise multiplication is applied over the output gate (Eq. 2.4) and a non-linear function ϕ (e.g., tanh function) over the candidate $\mathbf{c}^{(l)}[n]$ (Eq. 2.5).

$$\mathbf{f}^{(l)}[n] = \underbrace{\sigma\left(\mathbf{W}_f^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_f^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{forget gate}} \tag{2.2}$$

$$\mathbf{i}^{(l)}[n] = \underbrace{\sigma\left(\mathbf{W}_{\mathrm{i}}^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_{\mathrm{i}}^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{input gate}} \tag{2.3}$$

$$\mathbf{o}^{(l)}[n] = \underbrace{\sigma\left(\mathbf{W}_o^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_o^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{output gate}} \tag{2.4}$$

$$\mathbf{c}^{(l)}[n] = \underbrace{\mathbf{f}^{(l)}[n] \odot \mathbf{c}^{(l)}[n-1] + \mathbf{i}^{(l)}[n] \odot \underbrace{\phi\left(W_c \cdot \mathbf{h}^{(l-1)}[n] + V_c \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{V-RNN}}}_{\text{candidate}} \tag{2.5}$$

$$\mathbf{h}^{(l)}[n] = \underbrace{\mathbf{o}^{(l)}[n] \odot \phi\left(\mathbf{c}^{(l)}[n]\right)}_{\text{hideen state}} \tag{2.6}$$

Figure 2.2 shows the computational operations of a LSTM cell and its transition steps, where $f[n], \mathrm{i}[n], o[n]$ are V-RNNs with a sigmoid activation functions, and $\hat{h}[n]$ is a V-RNN with a tanh activation function.

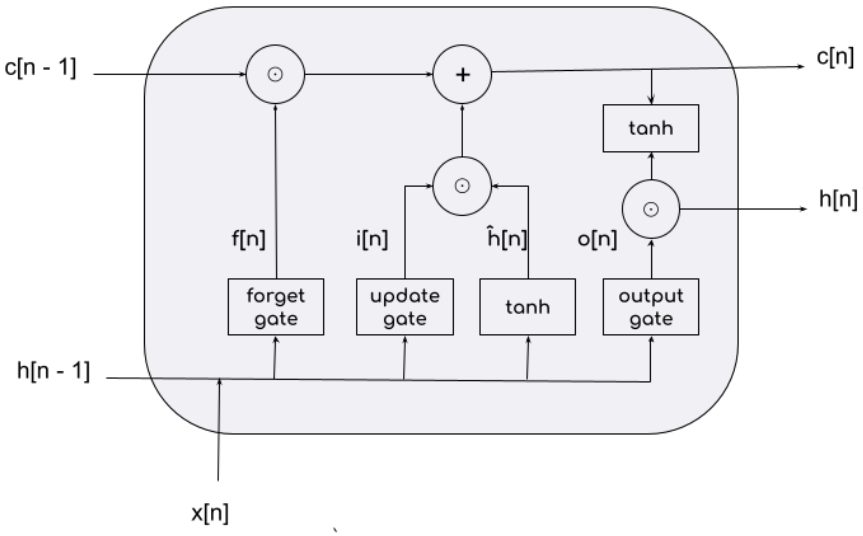


Figure 2.2: LSTM recurrent cell.

2.1.3 Gated Recurrent Unit (GRU)

GRU can be viewed as a simplification of the LSTM, that only uses *update* and *reset* gates to achieve the same goal [2]. The reset gate (Equation 2.7) decides how much of the information from the previous time-stamp to pass to the candidate for the next step (Equation 2.8). The update gate (Equation 2.9) controls how much of the information from the previous time-stamp and the candidate to pass to the next step.

$$\mathbf{r}^{(l)}[n] = \underbrace{\sigma\left(\mathbf{W}_r^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_r^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{reset gate}} \tag{2.7}$$

$$\hat{\mathbf{h}}^{(l)}[n] = \underbrace{\phi\left(\mathbf{W}_{\hat{h}}^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_{\hat{h}}^{(l)} \cdot \mathbf{h}^{(l)}[n-1] \odot \mathbf{r}^{(l)}[n]\right)}_{\text{candidate}} \tag{2.8}$$

$$\mathbf{z}^{(l)}[n] = \underbrace{\sigma\left(\mathbf{W}_z^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}_z^{(l)} \cdot \mathbf{h}^{(l)}[n-1]\right)}_{\text{update gate}} \tag{2.9}$$

$$\mathbf{h}^{(l)}[n] = \underbrace{\left(1 - \mathbf{z}^{(l)}[n]\right) \odot \hat{\mathbf{h}}^{(l)}[n] + \mathbf{z}^{(l)}[n] \odot \mathbf{h}^{(l)}[n-1]}_{\text{hideen state}} \tag{2.10}$$

Figure 2.3 describes the computational operations of a GRU cell and their transition steps, where $r[n]$ and $z[n]$ are V-RNNs with a sigmoid activation function, and $\hat{h}[n]$ is a V-RNN with a tanh activation function.

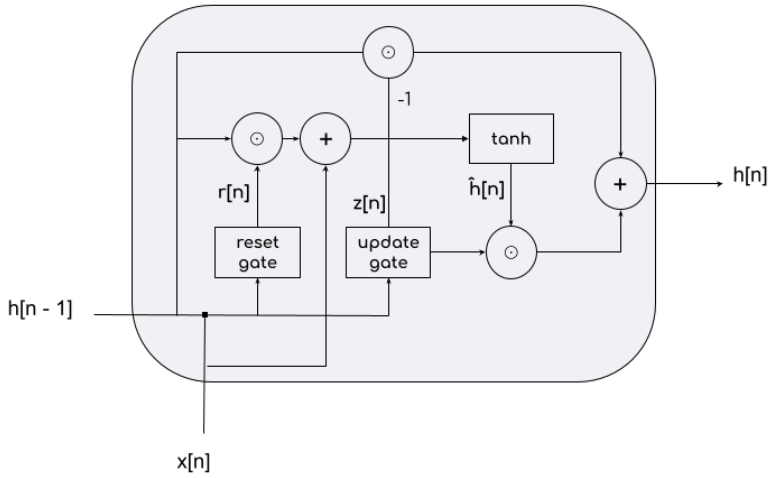


Figure 2.3: GRU recurrent cell.

2.1.4 RNN Architectures

The above-mentioned RNN cells can be used to build many architectures to solve different tasks. For example, there is the Many-to-Many architecture, which consists of processing an input sequence and producing an output sequence (commonly used for machine translation). Another architecture is the Many-to-One architecture, which takes a sequence and produces a single step (commonly used for classification tasks). There is also the One-to-Many architecture, which takes a single time step and produces an output sequence (commonly used for music generation) [23]. In addition, these RNN cells can also be stacked to build a deep RNN [73].

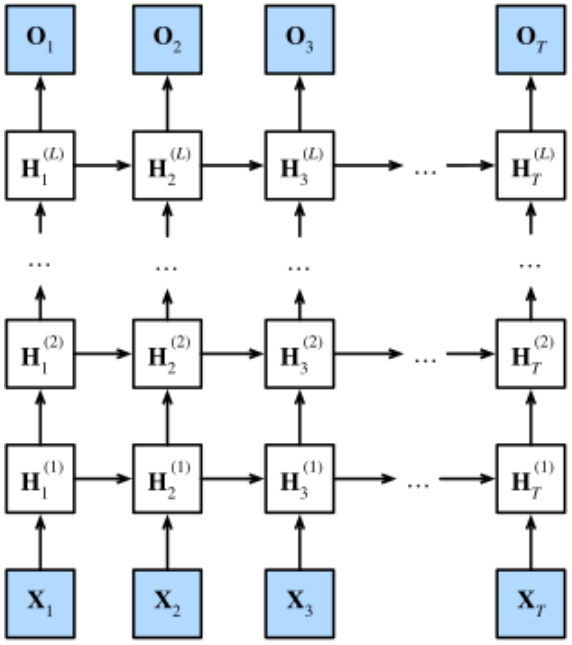


Figure 2.4: Deep and Many to many RNN architecture (space and time). Taken from [73] as an open source resource.

2.2 Spiking Recurrent Neural Networks (SRNNs)

A particularly simple model of a spiking neuron is the *Leaky Integrate-and-Fire* (LIF) model [20]. Its neural signal consists of voltage pulses called *action potentials* or *spikes*. The information is encoded in the presence or absence of a spike, where spikes are reduced to *events* that happen at a moment in time. Integrate-and-fire models are composed of an equation that describes the evolution of the membrane potential[1] and a mechanism to generate spikes. Thus, their neural dynamics can be seen as a summation or integration process combined with a mechanism that triggers spikes when the voltage (containing the summed effect of all inputs) is above some threshold [20].

In order to align the LIF model with deep learning principles, Neftci et al. [49] define a LIF neuron in layer l in a differential form as follows:

$$\tau_{mem}\frac{\mathrm{d}\mathbf{u}^{(l)}(t)}{\mathrm{d}t} = -\left(\mathbf{u}^{(l)}(t) - u_{rest}\right) + R\mathbf{i}^{(l)}(t), \tag{2.11}$$

where $\mathbf{u}^{(l)}(t)$ is the membrane potential, u_{rest} is the resting potential, τ_{mem} is the membrane time constant, R is the input resistance, and $\mathbf{i}^{(l)}(t)$ is the input current.

In addition, Neftci et al. [49] assume that synaptic currents sum linearly as follows:

$$\frac{\mathrm{d}\mathbf{i}^{(l)}(t)}{\mathrm{d}t} = -\frac{\mathbf{i}^{(l)}(t)}{\tau_{\text{syn}}} + \underbrace{\mathbf{W}^{(l)} \cdot \mathbf{h}^{(l-1)}(t)}_{\text{feed forward}} + \underbrace{\mathbf{V}^{(l)} \cdot \mathbf{h}^{(l)}(t)}_{\text{recurrent}}, \tag{2.12}$$

where τ_{syn} is the synaptic time constant. In this equation, a linear transformation between the previous layer activation and the current layer weights $\mathbf{W}^{(l)}$ is applied. Further, another linear transformation between $\mathbf{V}^{(l)}$ and the current layer activation is applied. The initial conditions of both equations, 2.11 and 2.12, change instantaneously when a spike occurs.

A reset term is added to Eq. 2.11 to instantaneously decrease the membrane potential (by amount $u_{rest} - \vartheta$) whenever the neuron emits a spike as follows:

$$\tau_{mem}\frac{\mathrm{d}\mathbf{u}^{(l)}(t)}{\mathrm{d}t} = -\left(\mathbf{u}^{(l)}(t) - u_{rest}\right) + R\mathbf{i}^{(l)}(t) + (u_{rest} - \vartheta)(\mathbf{h}^{(l)}(t)), \tag{2.13}$$

where $\mathbf{h}^{(l)}(t)$ is a vector of spikes. In discrete time, the vector of spikes in layer l at time step n is denoted by $\mathbf{h}^{(l)}[n]$, which is expressed as a nonlinear function applied over the difference between the membrane potential and a threshold, $\mathbf{h}^{(l)}[n] = \Theta(\mathbf{u}^{(l)}[n] - \vartheta)$, where Θ is the Heaviside step function and ϑ corresponds to the firing threshold.

In the following subsections we define different SRNN variants produced for mapping LIF neurons in discrete time. These models are based on different assumptions, resulting in different formulations.

[1] It is the difference in electrical potential between the interior and the exterior of a biological cell membrane [35].

2.2.1 Vanilla SRNN (V-SRNN)

We call Vanilla SRNN to the result of mapping LIF neurons in discrete time. This mapping has some assumptions for the SRNN formulation. First, the input current does not have its own dynamics[2] (Eq. 2.14). Second, the resting potential (u_{rest}) is defined using a scalar value different from zero, the input resistance (R) is equal to 1, and the time membrane constant and the step simulation are contained in $\beta = \frac{\delta t}{\tau_{mem}}$, according to the Eq. 2.15, which is obtained by an algebraic manipulation of the LIF Euler approximation. This approximation decouples the membrane potential into two parts: *reset term* and *input current*. The reset term works as a leaky factor of the input current.

$$\mathbf{i}^{(l)}[n] = \underbrace{\mathbf{W}^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}^{(l)} \cdot \mathbf{h}^{(l)}[n-1]}_{\text{input current}} \tag{2.14}$$

$$\mathbf{u}^{(l)}[n] = \underbrace{\underbrace{(1-\beta)\left((1-\mathbf{h}^{(l)}[n-1]) \odot \mathbf{u}^{(l)}[n-1] + \mathbf{h}^{(l)}[n-1]u_{rest}\right)}_{\text{leak}} + \beta u_{rest} + \beta \mathbf{i}^{(l)}[n]}_{\text{membrane potential}} \tag{2.15}$$

Equation 2.16 defines a spike event $\mathbf{h}^{(l)}[n]$ in layer l and time step n as follows:

$$\mathbf{h}^{(l)}[n] = \Theta(\mathbf{u}^{(l)}[n] - \vartheta), \tag{2.16}$$

where in order to produce a spike event, a step function Θ is applied to the difference between the membrane potential $\mathbf{u}^{(l)}[n]$ and a threshold voltage value ϑ. This threshold works as a barrier that, when overcome, allows the neuron to emit a spike.

Figure 2.5 shows the computational operations of a V-SRNN cell and its transition steps, where i[n] could be considered a V-RNN with an *identity* activation function.

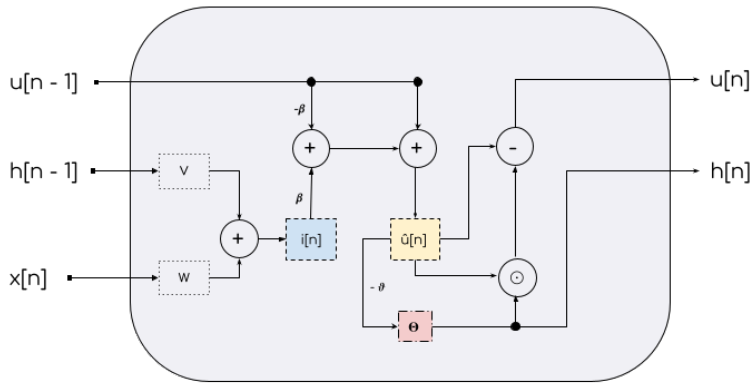


Figure 2.5: Vanilla spiking recurrent cell.

[2]It means that the initial conditions do not change when a spike occurs.

2.2.2 Cramer SRNN (C-SRNN)

The formulation by Cramer et al. [14] and Neftci et al. [49] are similar to one another. First, both assume that input current has its own dynamics as follows:

$$\mathbf{i}^{(l)}[n] = \underbrace{\alpha\mathbf{i}^{(l)}[n-1] + \mathbf{W}^{(l)} \cdot \mathbf{h}^{(l-1)}[n] + \mathbf{V}^{(l)} \cdot \mathbf{h}^{(l)}[n-1]}_{\text{input current}}, \tag{2.17}$$

where $\alpha = \frac{\mathrm{d}t}{\tau_{syn}}$.

Second, the threshold ϑ is equal to 1 and the resting potential u_{rest} is equal to 0. However, both formulations differ in their resistance value R, where Cramer et al. [14] use a value of $1 - \lambda = 0.025$ for scaling the input current, while Neftci et al. [49] set it to 1.

A more simple formulation of membrane potential, previously expressed by the Eq. 2.15, is defined as follows:

$$\underbrace{\mathbf{u}^{(l)}[n] = \underbrace{(1-\beta)\left((1-\mathbf{h}^{(l)}[n-1]) \odot \mathbf{u}^{(l)}[n-1]\right)}_{\text{leak}} + \beta\mathbf{i}^{(l)}[n]}_{\text{membrane potential}}. \tag{2.18}$$

Simplicity comes from ignoring u_{rest} and reducing the LIF approximation. By ignoring u_{rest}, the dynamics of this model are reset to zero when a spike occurs.

Figure 2.6 shows the computational operations of a C-SRNN cell and its transition steps, where i[n] could be considered a V-RNN with an *identity* activation function plus the previous hidden state.

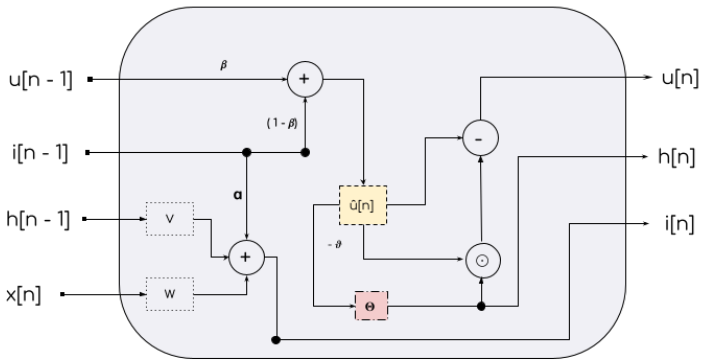


Figure 2.6: Cramer spiking recurrent cell.

2.2.3 Adaptive SRNN (A-SRNN)

A-SRNN is based on almost all assumptions and equations of the V-SRNN. It only differs in the threshold definition, because it evolves over time producing its own dynamics [4]. The threshold dynamics is represented as:

$$\boldsymbol{\eta}^{(l)}[n] = \rho\boldsymbol{\eta}^{(l)}[n-1] + (1-\rho)\mathbf{h}^{(l)}[n-1], \tag{2.19}$$

where $\rho = \frac{\delta t}{\tau_{adapt}}$ is a scaling factor and $\mathbf{h}^{(l)}[n-1]$ is the previous hidden state. This mechanism produces small changes when a spike event occurs.

The adaptive threshold is computed as:

$$\boldsymbol{\vartheta}^{(l)}[n] = \vartheta_0 + B\boldsymbol{\eta}^{(l)}[n], \tag{2.20}$$

where B is a scaling factor and ϑ_0 is the initial threshold. This threshold is then used in Eq. 2.16.

Figure 2.7 shows the computational operations of an Adaptive Spiking Recurrent Cell and its transition steps, including the $\boldsymbol{\eta}$ computation.

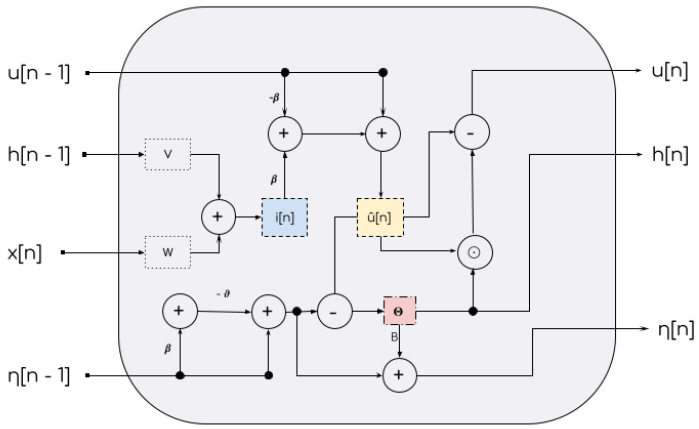


Figure 2.7: Adaptive spiking recurrent cell.

2.3 Supervised Learning

Supervised learning is a machine learning paradigm for acquiring the relationship information of a set of labelled training data. Its goal is to learn the mapping between the input and the output to then predict the output of new inputs. These outputs can be discrete or continuous values, that is known as classification and regression tasks, respectively [43].

In supervised learning, it is necessary to adapt the parameters or weights of the model, such that predictions $\hat{y}$ come close to the ground truth y. Backpropagation is a learning rule widely used for this purpose [68]. In addition, an optimisation criterion is necessary (or loss function) to quantify the performance of the model in order to maximise it (or minimise the error).

2.3.1 Cross-Entropy Optimisation Criterion

The Squared Error (SE) and the Cross-Entropy (CE) criteria are the most popular choices in state-of-the art implementations. An ANN can be trained by minimising either function, as long as it is capable of approximating the true posterior distribution arbitrarily close [22]. The CE method measures the Kullback Leibler distance between two distributions $P = \{p_1, p_2, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_n\}$ as follows [40]:

$$D(Q, P) = \sum_{k=1}^{n} q_k \log_2 \frac{q_k}{p_k}. \tag{2.21}$$

For multiclass classification, the loss of each class given an input vector is summed as follows:

$$CrossEntropy = -\sum_{c=1}^{C} y_c \log(p_c), \tag{2.22}$$

where C is the number of classes, y is the ground truth and p_c is the predicted probability of belonging to class c.

2.3.2 Evaluation Metric: Accuracy

The performance of a trained model is measured on a test set (different from training), where the predicted labels are compared with the actual labels. The *confusion matrix* shown in Table 2.1 is a standard way to display the classification results.

Table 2.1: Confusion matrix for a binary classification.

		Predicted class	
		1	*0*
Actual class	*1*	True Positives (TP)	False Negatives (FN)
	0	False Positives (FP)	True Negatives (TN)

Accuracy

Accuracy is the proportion of correct classifications. It is computed as the number of instances correctly classified divided by the total number of instances evaluated.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{2.23}$$

2.4 Training RNNs

Neural network training is based on forward and backward propagation. Forward propagation calculates all variables traversing the computational graph in the direction of the dependencies. Then, based on the chain rule, backpropagation computes and stores the gradients [73]. Backpropagation calculates the derivatives of a single target quantity (such as the classification error) with respect to the parameters or weights of a classification rule [68].

Backpropagation extended to sequence models like RNNs is known as *Backpropagation Through Time*. It allows the calculation of derivatives to optimise an iterative procedure in time [68]. The hidden states and outputs at each time step are computed as:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}, w_h), \\ o_t &= g(h_t, w_o), \end{aligned} \tag{2.24}$$

where f and g are transformation functions. In this recurrent computation, h_t depends on both h_{t-1} and w_h, and the computation of h_{t-1} also depends on w_h [73].

The objective function L measures the discrepancy (or loss) between the output o_t and the desired label y_t and it is computed as:

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^{T} l(y_t, o_t), \tag{2.25}$$

where L is the loss function, y_t is the label of the class, o_t is the output of the network and T is the max time of the sequence. The chain rule cleared from Eq. 2.25 is clearly exposed, where the time dependency and backward error propagation for two paths is expressed as:

$$\frac{\partial L}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial g(h_t, w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h} \tag{2.26}$$

According that the loss in Eq. 2.26 computes the gradients with regard to the parameters w_h using the chain rule. The first path error propagation arises from the last output of the network and the second one arises in the previous hidden step. Following the dependencies, the recursive computation of $\partial h_t / \partial w_h$ is given by [73]:

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h} \tag{2.27}$$

2.4.1 Vanishing and Exploding Gradient Problem

Dependency learning with gradient descent based algorithms becomes more difficult as the duration of the dependencies to be captured increases [5], because the gradients tend to explode or vanish. In RNNs, the exploding or vanishing gradient becomes severe, because the models are not only deep in terms of time, but also deep in terms of space (number of layers).

These gradient problems are related to the network architecture and the chain rule of gradient. Since, chain rule is a formula for calculating the derivative of a composite function, sometimes via multiplying process. This process leads to two problems. The first one is related to the explosion of the gradient that is produced by larger gradient values. The second one is related to the vanishing of the gradient that is produced by small gradient values (close to zero).

2.4.2 Gradient Clipping

Gradient clipping is a technique to avoid gradient explosion. According to [23], strongly non-linear functions, such as those computed by a recurrent network over many time steps, tend to have derivatives that may be of very large or very small magnitude. In reference to large gradient magnitudes one type of simple solution that has been used by machine learning practitioners is *gradient clipping* [23]. As shown in the Figure 2.8, when the parameter update (w, b) has the same direction as the true gradient, with the clipping of the gradient norm, the norm of the parameter update vector is now constrained.

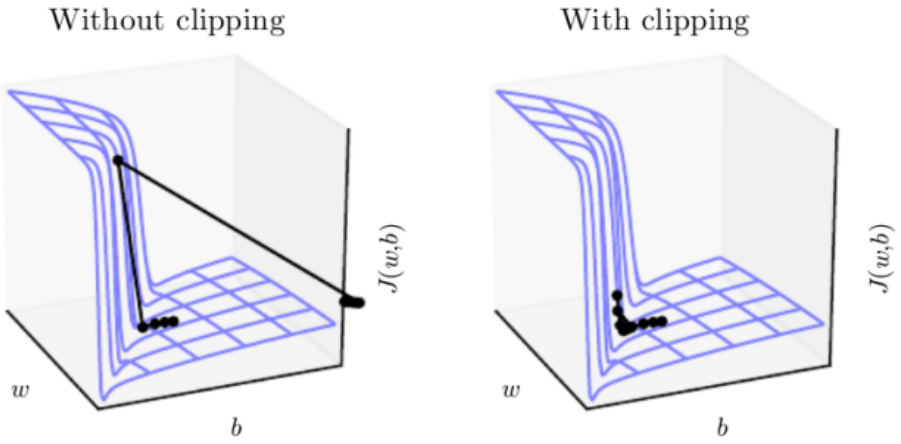


Figure 2.8: Effect of gradient clipping on a recurrent network with w and b parameters. Taken from [23]. The figure shows a landscape of the loss function considering two scenarios. The figure on the left shows that the gradients are very large which creates a problem when trying to reach the optimal solution. The figure on the right shows the effect of gradient clipping which cuts off the large gradients to reach the optimal solution.

2.4.3 Reducing the Learning Rate at the Plateau

Reducing the learning rate at the plateau is a technique that helps the model to continue learning once learning stagnates. It was proposed as a solution for the saddle points (Figure 2.9) and local minimum problem of the loss function [9], [23]. The initial selection of the learning rate is usually arbitrary, which does not guarantee that it is the optimal. Therefore, the use of a learning rate that decreases when the network stops learning is a technique that could be very useful in this work.

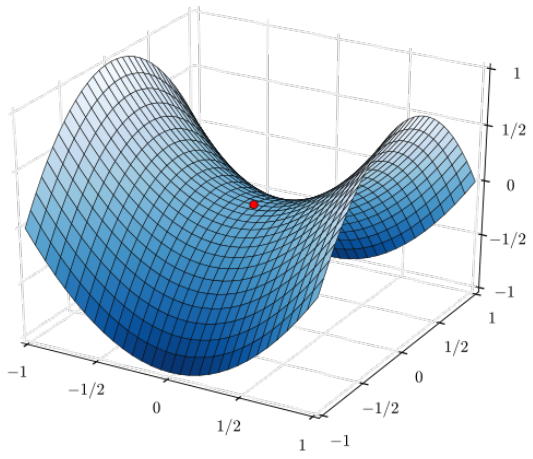


Figure 2.9: Saddle point in loss landscape. The red point gives an illusion that the model has converged to a minimum. Although there is a minimum on the x-axis, there is also a local maximum in another direction, so if the contour is flatter towards the x-direction, the learning rule would still oscillate in the y-axis direction.

2.5 Information Theory

Information theory is the field that studies the quantification, storage and communication of digital information [56]. It is used to answer two fundamentals questions in communication theory: (1) what is the ultimate data compression and (2) what is the ultimate transmission rate of communication [13]. Information, which is not a physical entity but an abstract concept, is hard to quantify in general [18]. Nevertheless, there are several quantities to measure the information in a random variable (e.g., the entropy and mutual information). In reference to the optimisation criteria using information theory criteria, we introduce information bottleneck based on [6] and [64].

2.5.1 Random Variables

A random variable is a variable whose possible values are numerical outcomes of a random phenomenon. Commonly random variables are denoted with capital letters, e.g., $X : S \to \mathbb{R}$. Informally, a random variable assigns numbers to the outcomes in the sample space S. So, instead of focusing on the outcomes themselves, we highlight a specific characteristic of the outcomes [17], [62].

2.5.2 Entropy

Entropy H is defined as the average level of information associated to any random variable. Specifically, it is a measure of uncertainty of a random variable [13]. When the entropy is high, more uncertain is the random variable. In addition, it gives the fundamental limit for data compression. In terms of the quantities, entropy is the average number of bits needed to describe random variable X.

2.5.3 Mutual Information

The mutual information of two random variables X and Y (Figure 2.10), is represented as $I(X;Y)$ and indicates how much information these variables share based on the entropy [13], [18], [55]. It can be interpreted as a measure of the mutual dependence between the two random variables. In addition, this quantity allows the computation of the capacity of a channel.

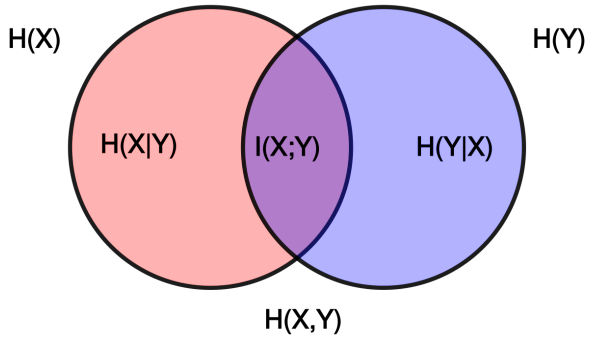


Figure 2.10: Mutual information between two random variables.

2.5.4 Information Bottleneck Principle

The Information Bottleneck (IB) method was introduced in [64] as an information theoretic principle. This method is commonly used as a data compressor and as optimisation criterion. The objective of IB is to extract the relevant information contained in an input random variable X about an output random variable Y [63].

Formally, The goal of the IB method is to construct a random variable $\widehat{X}$, which tries to keep the information that X has over Y via a stochastic mapping defined by the conditional $p(\widehat{X}|X)$ [6].

$$L_{IB} = I(\widehat{X};Y) - \gamma I(\widehat{X};X) \tag{2.28}$$

L_{IB} Measures how informative the compressed representation $\widehat{X}$ is about Y. The second term is a Lagrange multiplier, which penalises complex representations of $\widehat{X}$ and can be regarded as a information regularization term. The IB method consists in finding a conditional

probability distribution $p(\widehat{X}|X)$ that maximises L_{IB} under the condition that $\widehat{X}$, X and Y form the Markov Chain $Y \to X \to \widehat{X}$, the γ parameter determines the degree of compression via the trade off between the relevant information that $\widehat{X}$ carries about Y and complexity of $\widehat{X}$ [6].

Chapter 3

Methodology

This work formulates a new variant of SRNN based on deep learning models. Additionally, we focus on evaluating the performance, optimisation criterion, and time resolution of the most common SRNNs and traditional RNNs variants.

In this chapter, we describe the data used in our experiments and their pipeline pre-processing. Next, we explain the experimental framework, including the elements used to evaluate the SRNN and RNN arquitectures in two supervised learning tasks. Then, we describe the formulation of a new SRNN variant. In the following section, we categorise and model the memory in SRNNs. Finally, we formulate the adaptation of the information bottleneck as an optimisation criterion in the SRNNs models.

3.1 Experimental Data

Our datasets consist in binary sequences for classification and sequence labeling tasks. The first one corresponds to the parity bit problem (also known as the Theoretically Sequential XOR problem), which is used to explore the properties of the RNNs models [38]. The second one is the Spike Heidelberg Dataset, which is used to evaluate the performance of both SRNNs and RNNs [14].

3.1.1 Parity Bit Problem or Sequential XOR

The parity bit is used for error communication decoding [76]. It is also used as a synthetic dataset for machine learning in classification and sequential labeling tasks [16], [33], [38]. In the classification task, the target of this problem can be computed via a XOR sum of the bits defined as follows:

$$y = \sum_{t}^{n} x_t \pmod 2, \tag{3.1}$$

where x_t means the binary value at a time step t, and n means the size of the sequence. In a sequential labeling task, the target is also sequential, and the model needs to learn if at a

certain time t, the sequence of bits is even or not. This can be expressed as:

$$y_{\mathrm{i}} = \sum_{t}^{\mathrm{i}} x_t \pmod 2. \tag{3.2}$$

The Equation 3.2 defines the dataset for this task, which we use to conduct our experiments.

Since the Eq. 3.2 depends on i, it is possible to generate different sequence lengths. Different sequence lengths make the problem more complex to solve, because as i increases, the generation of unique sequences grows exponentially as 2^i. This exponential growth of unique elements ensures that the model can not memorise the data and needs to learn the function that generates it.

We generated four versions of this dataset as a result of varying the sequence lengths from 16 to 128 time steps (Fig. 4.1). Table 3.1 specifies the sequence size and the number of unique sequence samples for each version. In addition, this table details the number of samples used for training and testing the models.

Table 3.1: List of Parity Bit versions used in our experiments.

Version	Sequence size	Unique samples	Train samples	Test samples
1	16	2^{16}	10^3	10^2
2	32	2^{32}	10^4	10^3
3	64	2^{64}	10^5	10^3
4	128	2^{128}	10^5	10^3

3.1.2 Spiking Heidelberg Digits (SHD)

In [14], two novel spike-based classification datasets were proposed, following a general audio-to-spiking conversion procedure. These datasets are called *Spiking Speech Command (SSC)* and *Spiking Heidelberg Digits (SHD)*. In this thesis, we use the SHD dataset because it allows us to perform several experiments due to the number of samples is lower than SSC dataset. SHD dataset is based on a spoken digit dataset recorded by the University of Heidelberg.

The Heidelberg Digits (HD) audio dataset contains 20 classes of spoken digits, namely the English and German digits from zero to nine, spoken by 12 speakers. The recording time of the samples on average is around 750ms with +/- 500ms variance. This variance causes some problems for mini-batch training due to the different duration of the samples. To solve these problems, we define the following rules:

- To cut the audio sample if this is larger than one second (Simulation duration).
- To fill with zeros at the beginning of the sample until completing one second (Zero Padding).

Using the defined simulation duration (one second) produces 10k time steps because the SHD was generated using a 0.1ms simulation step size. The use of this simulation step is computationally demanding, so we use binning to generate two reduced version of it. These versions were generated using 4ms and 10ms simulation step sizes.

3.2 Experimental Framework

In this section, the SRNNs and traditional RNNs are defined as directed graphs. These models are configured for supervised learning. Next, these models are built as multilayer networks by varying the hyperparameters of the architecture. Finally, these architectures are trained using strategies to avoid problems with gradients. All previously mentioned architectures were implemented using the *Pytorch* library [51].

The experimental framework consists of a series of steps and methods listed in the below sections. The results obtained with this framework are considered as baselines to compare our results with existing works and with the new SRNN proposed in this work.

3.2.1 Recurrent Neural Networks Graph Representation

To analyse the dependency and information flow in SRNNs and traditional RNNs, we created directed computational graphs. These directed graphs allow us to identify which edge will be adjusted and which nodes will be used to solve a problem. For simplicity, these graphs were designed to represent only neural networks with a single neuron in a hidden layer.

Figure 3.1 represents the computational graph for a Vanilla Recurrent Neural Network. This graph has only two nodes and two edges with their respective weights. One of the nodes is the hidden state which has its own loop and the other node is the input data connected to the hidden state. This graph also can represent any traditional RNNs and SRNNs if the parameters W, V are equal to one. In addition, this graph provides a reference to compare the other models in terms of information flow and structure.

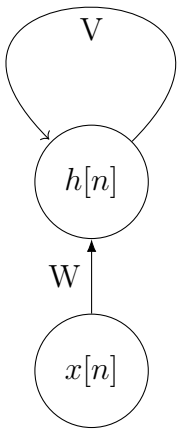


Figure 3.1: V-RNN computational graph, where $x[n]$ is the input signal, $h[n]$ is the hidden signal at time step n, and W and V are the learnable or free parameters.

In order to understand how information flows in the GRU model, we designed a directed graph, as shown in Figure 3.2. This graph has more weighted edges than V-RNN graph (five nodes and ten edges) and summarise the Eq. 2.7, 2.8, 2.9. This graph shows that the hidden state depends on the update gate $z[n]$ and a V-RNN as a candidate $\hat{h}[n]$. The candidate also depends on the reset gate $r[n]$, which defines how much information to consider from

the previous hidden state. These mechanisms ($z[n]$, $r[n]$, $\hat{h}[n]$) need two edges with learnable parameters for working.

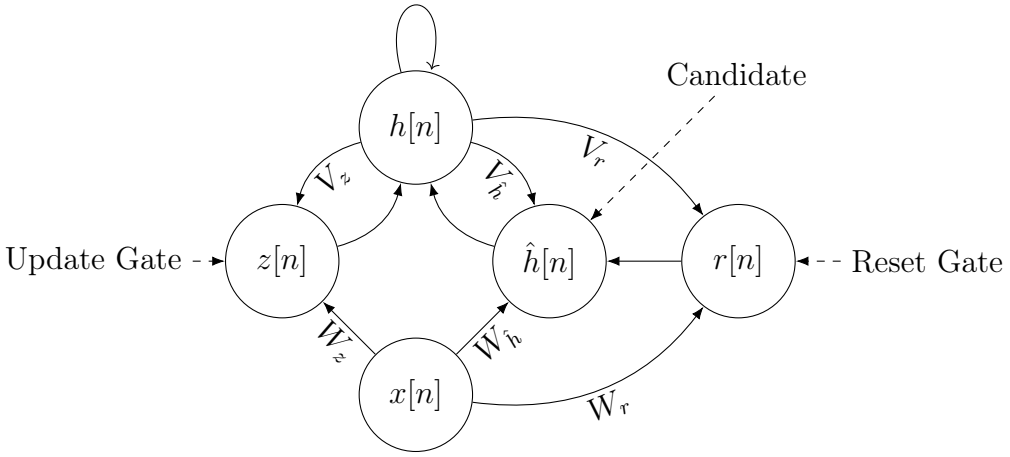


Figure 3.2: GRU computational graph, where $x[n]$ is the input, $h[n]$ is the hidden state, $z[n]$ is the update gate, $\hat{h}[n]$ is the candidate gate, $r[n]$ is the reset gate in the time step n, and W and V are the learnable or free parameters.

The information flow in the LSTM model, as shown in 3.3, is more complex than those of the two previous models. It adds more parameters for including an the output gate that controls how much information is passed to the hidden state. This hidden state $h[n]$ depends on a candidate node, which has a self-loop with no parameters. The candidate node $\hat{h}[n]$ adds an external memory mechanism, which is discarded when the forget gate is completely deactivated ($f[n] = 0$). In some cases, the candidate is completely deactivated when the input and forget gate are equal to zero ($f[n] = 0$ and $\mathrm{i}[n] = 0$).

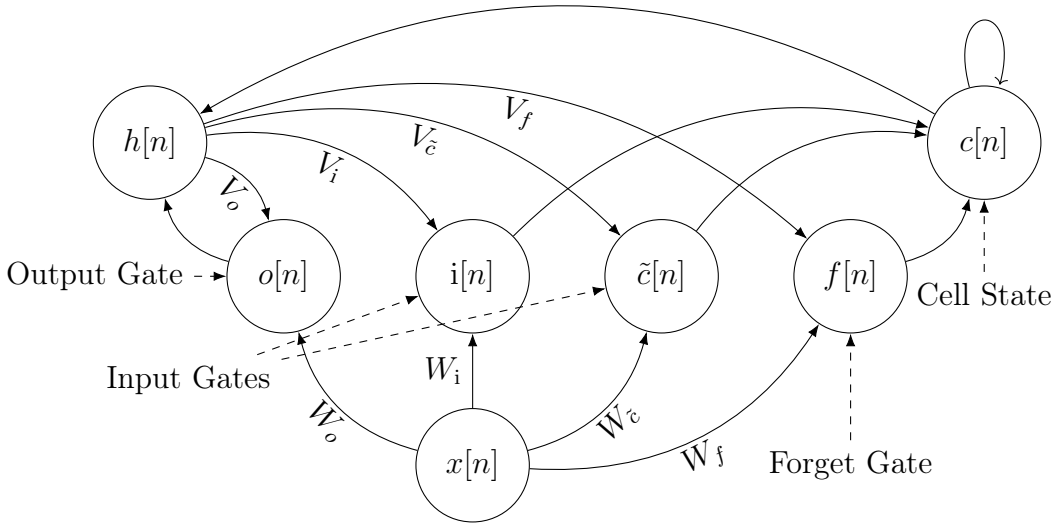


Figure 3.3: LSTM computational graph, where $x[n]$ is the input, $h[n]$ is the hidden state, $o[n]$ is the output gate, $\hat{c}[n]$ is the candidate gate, $f[n]$ is the forget gate, $\mathrm{i}[n]$ is the input gate in the time step n, and W and V are the learnable or free parameters.

Figure 3.4 shows how the information flows in a Vanilla Spiking Recurrent Neural Network, using only four nodes and six edges. In addition, it displays some similarities with the GRU and LSTM graphs, considering that the node $u[n]$ has a self-loop with no parameters. The first notable difference with V-RNN is that it has more dependencies. However, both models have two edges with learnable parameters.

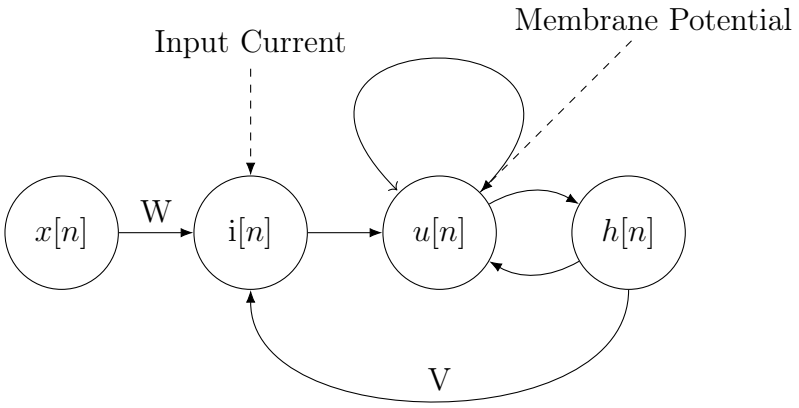


Figure 3.4: V-SRNN computational graph, where $x[n]$ is the input signal, i$[n]$ is the input current, $u[n]$ is the membrane potential, $h[n]$ is the spike event or hidden state signal, and W and V are the learnable or free parameters.

The information flow in the Crammer SRNN model is described in the Figure 3.5. It differs with V-SRNN only in the input current i$[n]$, which has a self-loop giving the model the ability to generate larger membrane potential voltages than the V-SRNN. This input node is larger because the previous i$[1, 2, \ldots, n]$ are added directly to the current input as expressed in the Equation 2.18.

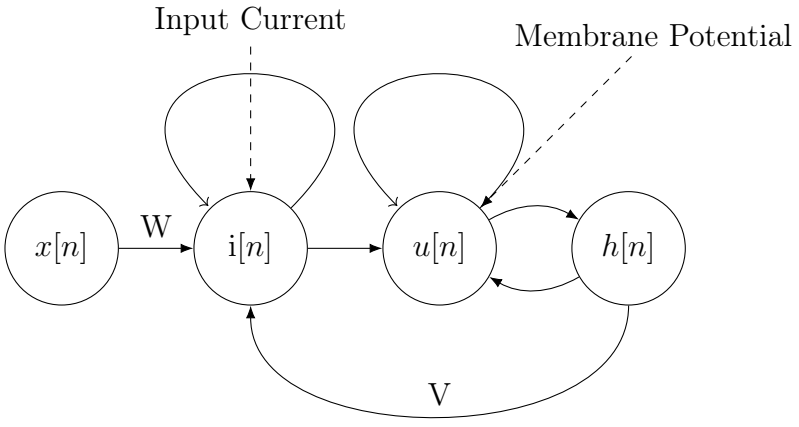


Figure 3.5: C-SRNN computational graph, where $x[n]$ is the input signal, i$[n]$ is the input current signal with its own loop, $u[n]$ is the membrane potential, $h[n]$ is the spike event or hidden state signal, and W and V are the learnable or free parameters.

The information flow in Adaptive SRNN is described in the Figure 3.6, which needs more nodes and edges compared to the V-SRNN. This structure has three nodes that feed back to

each other, where two of them have self-loops. In addition, the hidden state and the input data $x[n]$ are connected to the input current $\mathrm{i}[n]$ through two edges with learnable parameters. This structure also provides information on how the threshold is adapted. The adaptation is produced by $\eta[n]$ node, which adds new information when a spike occurs (Equation 2.19).

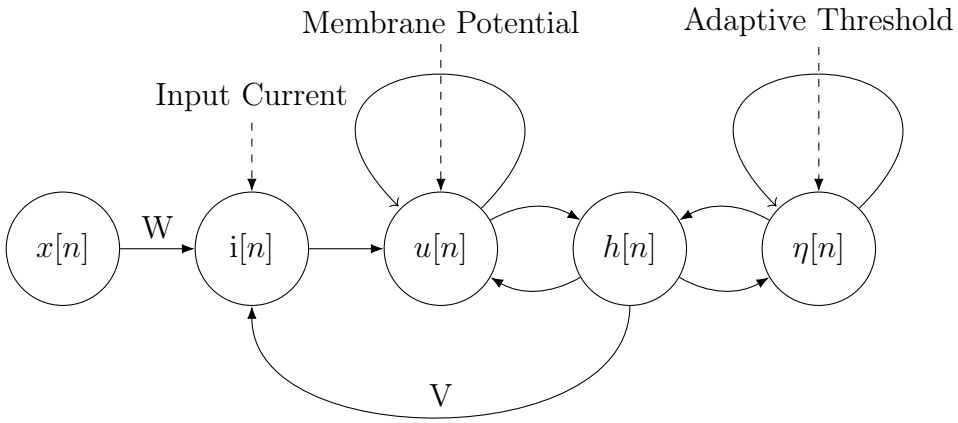


Figure 3.6: A-SRNN computational graph, where $x[n]$ is the input signal, $\mathrm{i}[n]$ is the input current signal, $u[n]$ is the membrane potential, $h[n]$ is the spike event or hidden state signal, $\eta[n]$ is the threshold dynamics, and W and V are the learnable parameters.

The above graphs share some similarities and differences in how information is conveyed. GRU, LSTM and V-SRNN have only one node with a self-loop without learning parameters, while the V-RNN, V-SRNN, C-SRNN and A-SRNN have two edges with two learnable parameters. In addition, C-SRNN and A-SRNN have two nodes with self-loops with no parameters on their edges.

All graphs have different paths where the information flows. However, in the SRNN models, the nodes $\mathrm{i}[n]$, $u[n]$, $\eta[n]$ and $h[n]$ can be grouped into a new node called $H[n]$ and considered as a V-RNN. Thus, the above grouping suggests that SRNN models have similar capabilities as V-RNN, which is a universal approximator of any dynamics, where one of the functions could be the SRNN formulations.

3.2.2 Supervised Learning setup for Multilayer SRNNs and traditional RNNs

To configure the SRNNs and traditional RNNs models for supervised learning, we selected a sequence labeling approach for working with both datasets. Regarding the **Parity Bit problem**, we used a single layered architecture as shown in Figure 3.7, according to the following expression:

$$\hat{\mathbf{y}}[n] = g(\mathbf{W}_y \cdot \mathrm{RNN}^{(L)}(\theta, H[n-1])), \tag{3.3}$$

where $\mathrm{RNN}^{(L)}$ is a function that could be any of the traditional RNNs or SRNNs, θ is the set of learnable parameters, H are the hidden states that evolve internally in the models, $\hat{\mathbf{y}}[n]$ is the output signal of the network at time step n, $\mathbf{y}[n]$ is the label in time step n and g is a

function that takes the output of the architecture and produces a valid probability density function. The output of the network is used to compute the cross entropy:

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}[n], \mathbf{y}[n]) = \mathbf{y}[n] \log \hat{\mathbf{y}}[n] + (1 - \mathbf{y}[n]) \log(1 - \hat{\mathbf{y}}[n]), \tag{3.4}$$

where $\mathbf{y}[n]$ is the target in time n. Finally, the cost function is defined as follows:

$$\mathcal{C} = \sum_{n}^{T} \mathcal{L}_{CE}(\hat{\mathbf{y}}[n], \mathbf{y}[n]) \tag{3.5}$$

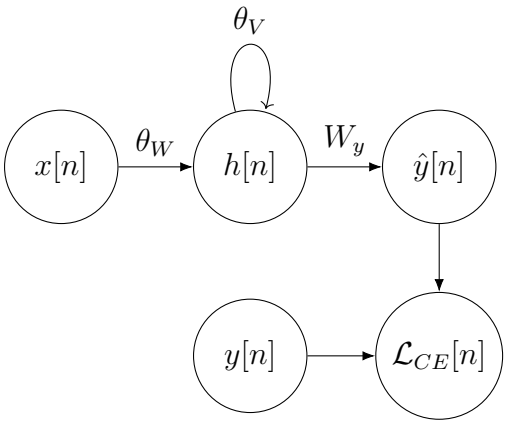


Figure 3.7: Sequence labeling for binary classification, where $x[n]$ is the input signal, $h[n]$ is the hidden signal, $\hat{y}[n]$ is the output of the network, $\mathcal{L}_{CE}[n]$ is the binary cross entropy, and $y[n]$ is the actual label at time n. This graph could represent the studied recurrent models, θ_W and θ_V are the free parameters.

Regarding the **Spiking Heidelberg Digits (SHD) problem**, we selected both multilayered and single-layered architectures as described in Fig. 3.8. These architectures follow the same parity bit formulation but with more stacked layers (Eq. 3.3). The last layer is linearly projected to produce the output of the architecture at a given time, $\hat{y}[n]$. This output is then reduced in time in order to generate a similar tensor to compare with the target, which is not sequential. Finally, a softmax layer is applied over $\hat{y}$ to produce a valid distribution. The following expressions describe how information flows to solve this problem:

$$\hat{\mathbf{y}} = S\left(\frac{\sum_{n}^{T} \hat{\mathbf{y}}[n]}{T}\right), \tag{3.6}$$

where S is the softmax function, T is the length of the sequence and $\hat{\mathbf{y}}$ is the predicted output of the network. Then, the output of the network is used to compute CE:

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{c}^{C} \mathbf{y}_c \log(\hat{\mathbf{y}}_c), \tag{3.7}$$

where C is the total number of classes. Finally, the cost function is defined in Eq. 3.8, where N is the number of samples in the batches. This function is similar to the Eq. 3.5, but

without time:

$$\mathcal{C} = \sum_{\mathrm{i}}^{N} \mathcal{L}_{CE}(\hat{y}_{\mathrm{i}}, y_{\mathrm{i}}) \tag{3.8}$$

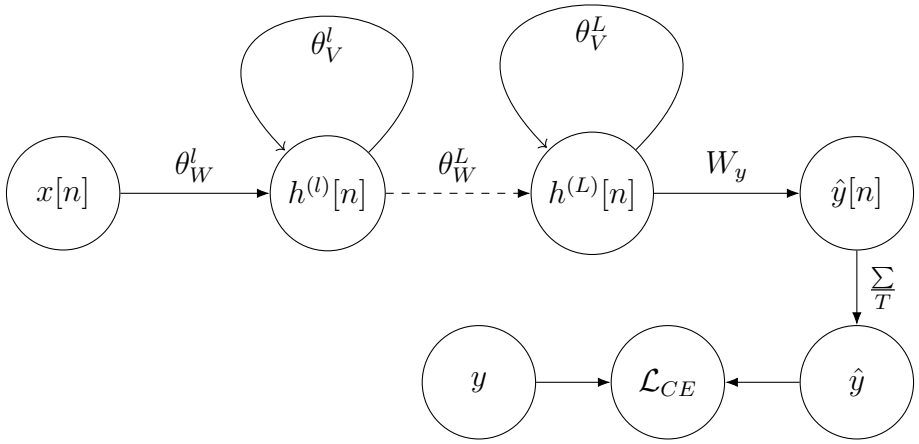


Figure 3.8: Sequence labeling for multiclass classification, where $x[n]$ is the input signal at time n, $h^{(l)}[n]$ is the hidden signal at time n and layer l, W_y are the learnable parameter for linear layer, $\hat{y}[n]$ is the output of the network at time n, $\hat{\mathbf{y}}$ is the reduced mean time prediction output class, $\mathcal{L}_{CE}$ is the cross entropy, and y stands for the actual labels. Since this graph represents any of RNN models, $\theta_W^{(l)}$ and $\theta_V^{(l)}$ are the free parameters.

As mentioned before, different architectures are explored for solving the parity bit and SHD problems. For the parity bit, we studied 8 architectures, while for the SHD, we studied 4 architectures. The Table 3.2 contains the studied architectures where *in* stands for the input features, the intermediate numbers represent the number of neurons in the hidden layers, and *out* stands for the output dimension of the architecture. For instance, the architecture *in-256-128-out* for the SHD problem has 2 recurrent layers, where the first layer has 256 hidden neurons and the second has 128 hidden neurons.

Parity Bit	**SHD**
Range from *in*-2^0-*out* to *in*-2^7-*out*	700-512-256-128-20
	700-256-128-20
where *in* and *out* are equal to 1	700-128-20
	700-512-20

Table 3.2: Architectures studied for parity bit and SHD dataset.

We defined these architectures based on what we wanted to test in the spiking models on the studied datasets. For example, in the case of SHD problem, we have two single hidden layer architectures and two multiple hidden layer architectures with different number of neurons. For the single hidden layer architecture of 128 neurons, we relied on the literature to have an evaluation benchmark. We also consider the single hidden layer architecture of 512 neurons to evaluate whether adding more neurons improves the results. Finally, we add

architectures with more than one hidden layer based on [23], which mentioned that greater depth seems to lead to better generalisation for a wide variety of tasks.

3.2.3 Backpropagation in SRNNs

The Backpropagation Through-Time algorithm (BPTT) for traditional RNNs is described in [23], [73]. In this section, we formalise the V-SRNN backpropagation algorithm based on abstract functions with their recurrence dependencies and learnable parameters. The following equations, from 3.9 to 3.12, describe the sequence configuration for a V-SRNN with a linear classification layer.

$$\mathbf{u}[n] = \mathbf{f}\left(\mathbf{u}[n-1],\ \mathbf{h}[n-1],\ \mathbf{W},\ \mathbf{V}\right) \tag{3.9}$$

$$\mathbf{h}[n] = \mathbf{\Theta}\left(\mathbf{u}[n],\ \vartheta\right) \tag{3.10}$$

$$\hat{\mathbf{y}}[n] = \mathbf{g}\left(\mathbf{h}[n],\ \mathbf{W}_y\right) \tag{3.11}$$

$$\mathcal{L} = \sum_{n}^{T} l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n]), \tag{3.12}$$

where $\mathbf{f}$ is a identity function, $\mathbf{\Theta}$ is a non-linear function, l is the loss function and $\mathcal{L}$ is the cost function.

The calculation of the gradient is divided into three main parts. The first part is about how $\mathbf{V}$ affects the loss function $l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])$. The second part considers the contributions of $\frac{\partial l(\hat{\mathbf{y}}[T],\ \mathbf{y}[T])}{\partial \mathbf{u}[T]}$, where T is the last step of the sequence. The final part computes the gradient of $\frac{\partial l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])}{\partial \mathbf{u}[n]}$ at any time.

To compute the gradient between the loss function and $\mathbf{V}$ (we omit the gradient computation of $\mathbf{W}$ because follows the same logic of $\mathbf{V}$) parameter ($\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$), it is necessary to add up all contributions which affect the loss function because $\mathbf{V}$ is present in all time steps. These contributions are decoupled using the chain rule. After that, the gradient of $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}$ is expressed as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{V}} = \frac{1}{T}\sum_{n=1}^{T} \frac{\partial l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])}{\partial \mathbf{u}[n]} \frac{\partial \mathbf{f}\left(\mathbf{u}[n-1],\ \mathbf{h}[n-1],\ \mathbf{W},\ \mathbf{V}\right)}{\partial \mathbf{V}}, \tag{3.13}$$

where the gradient of the membrane potential $\mathbf{u}[n]$ with respect to $\mathbf{V}$ is easy to compute. Similarly, the gradient of the loss function with respect of the membrane potential $\frac{\partial l(\hat{\mathbf{y}}[T],\ \mathbf{y}[T])}{\partial \mathbf{u}[T]}$ becomes easy to compute given that $\mathbf{u}[T]$ only appears in the last step (Equation 3.14).

$$\frac{\partial l(\hat{\mathbf{y}}[T],\ \mathbf{y}[T])}{\partial \mathbf{u}[T]} = \frac{\partial l(\hat{\mathbf{y}}[T],\ \mathbf{y}[T])}{\partial \hat{\mathbf{y}}[T]} \frac{\partial \hat{\mathbf{y}}[T]}{\partial \mathbf{h}[T]} \frac{\partial \mathbf{h}[T]}{\partial \mathbf{u}[T]} \tag{3.14}$$

Finally, the computation of the gradient with respect to the membrane potential in any time step $\frac{\partial l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])}{\partial \mathbf{u}[n]}$ is more complex than the previous gradients. This is because $\mathbf{u}[n]$

depends of $\mathbf{u}[n-1]$ and $\mathbf{h}[n-1]$. Nevertheless, this gradient can be computed in a recurrent way as described in the following expression:

$$\frac{\partial l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])}{\partial \mathbf{u}[n]} = \frac{\partial l(\hat{\mathbf{y}}[n],\ \mathbf{y}[n])}{\partial \hat{\mathbf{y}}[n]} \frac{\partial \mathbf{g}\left(\mathbf{h}[n],\ \mathbf{W}_y\right)}{\partial \mathbf{h}[n]} \frac{\partial \Theta(\mathbf{u}[n],\ \vartheta)}{\partial \mathbf{u}[n]} + \\ \frac{\partial l(\hat{\mathbf{y}}[n+1],\ \mathbf{y}[n+1])}{\partial \mathbf{u}[n+1]} \left\{ \frac{\partial \mathbf{u}[n+1]}{\partial \mathbf{u}[n]} + \frac{\partial \mathbf{u}[n+1]}{\partial \mathbf{h}[n]} \frac{\partial \Theta(\mathbf{u}[n],\ \vartheta)}{\partial \mathbf{u}[n]} \right\}, \quad (3.15)$$

where it can be clearly seen that there are two paths to calculate the membrane potential dependence. One of these paths has two branches, which are added up.

The step function Θ is commonly used to produce spike events in SRNNs. Its derivative is known as *impulse function* or δ *Dirac delta function*, which is not compatible with the BPTT learning rule. Several solutions have been explored for this problem [49]. In this work, we use the surrogate gradient proposed by [58]:

$$\frac{\mathrm{d}\Theta(\mathbf{u}[n])}{\mathrm{d}\mathbf{u}[n]} = \mathcal{N}(\mathbf{u}[n]|\vartheta, \sigma^2)\gamma, \quad (3.16)$$

where $\mathcal{N}$ is a normal distribution which depends of: threshold ϑ, $\mathbf{u}$[n] and a γ value. The γ value works as a scaling factor.

3.2.4 Training Setup

Training RNNs is a complicated task given their sequential nature. Therefore, several methods have been developed to train RNNs models. These methods partially avoid gradient problems and other phenomena improving the model performance and generalisation. The methods used in this work are detailed below (see Table 3.3).

The **Adam Learning Rule (AdamW)** method with weight decay is used to train SRNN and traditional RNN models. AdamW is one of the more practical ways of training RNNs because it improves their generalisation and avoids large values for weigths [44]. This learning rule is used from pytorch library implementation with their default setup.

The **Gradient Clipping** technique avoids the exploding gradient phenomenon when backpropagation is executed [75]. This technique requires a value defining the maximum allowable gradient for weight updating. This value is usually calculated using the L2-square of the gradients $|\nabla\theta|^2$.

An **Exponential Learning Rate** schedule is used for changing the learning rate when a validation metric does not improve [42]. The parameters for this method follow this setup: scaling factor of 2.5e5 with a patience of 5. Specifically, the ReduceLROnPlateau routine from pytorch is used.

The **Early Stopping** method allows halting the training task when a validation metric does not improve, allowing to evaluate more experiments and avoiding overfitting. This method is used from the pytorch library with a patience of 20 epochs to terminate training.

Table 3.3: Training setup for datasets.

Dataset	Epochs	Early Stopping	Learning Rate	Learning Rate Decay
Parity bit	10	2 epochs	$\frac{1}{n^l}$	0.10 scale factor
SHD	75	20 epochs	0.001	0.25 scale factor

3.2.5 Evaluation Procedure

As this work focuses on SRNNs and traditional RNNs for solving two classification tasks, we use the *accuracy* metric to define the effectiveness of the models. The second metric considered is the *number of parameters* of the model. The third metric is the average of number of spikes per layer. Finally, we also consider the *number of iterations* needed by the models to reach the best accuracy.

The parameters to be fitted in the models (weights) are randomly initialised. Therefore, we perform 5 runs to evaluate each experiment and compute their mean accuracy and variance. A statistical test is used to compare the performance between the models for solving the Parity Bit and SHD problems.

We conduct a hypothesis test to define which model is better than others. For that, there are several options such as t student test, Z test, permutation test, among others. In this work, we use the permutation test, which does not require the samples to be normal distributed and can be used for significance or hypothesis testing [65]. The probability, p is defined as:

$$p(t \geq t_0) = \frac{1}{(n+m)!} \sum_{j=1}^{(n+m)!} I(t_j \geq t_0), \tag{3.17}$$

where t_o is the observed value of the test statistic, and t is the statistic t-value computed from the resamples.

Measuring Spikes in SRNNs per Layer

To compute the spikes when the models reach a max accuracy, we define a mini batch sample size in time step n, as a random variable X_t. Then, we assume that X_t is a Bernoulli random variable, so we can compute the average probability of the positive outcomes $P = \sum_t^T \mathbb{E}[X_t]$. Finally, P is averaged in terms of time and number of validation examples. This metric, which considers the proportions of spikes, is called *number of spikes per layer* in the following chapters.

3.2.6 Final experimental flow

Once all the elements have been described, Figure 3.9 shows the steps to be followed to define a baseline. This baseline is used to compare the improvements to be made in the spiking models introduced in the following sections. These improvements are related to the calculation of parameters, adding external memory and adding information bottleneck.

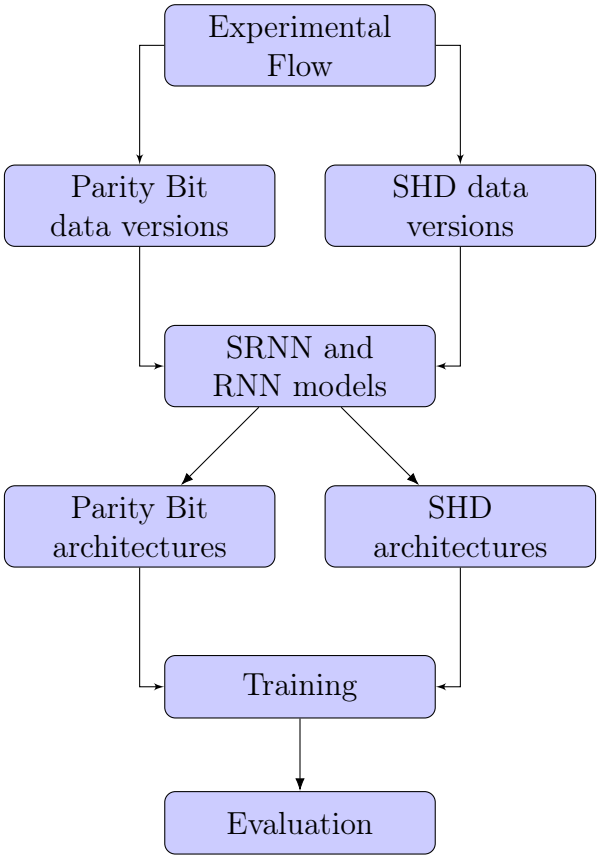


Figure 3.9: Experimental flow pipeline.

Experimental Framework Routine Implementation

We implement our proposed experimental framework (Fig. 3.9) using Python 3.7, PyTorch library and PyTorch Lightning, using Object Oriented Programming (OOP) and Functional Programming (FP) paradigms. This project can be found in the following repository: `https://github.com/jthoth/srnn-builder`.

3.3 SRNN Model Parameters

Setting the spiking model parameters β, u_{rest}, and ϑ from equations 2.15, 2.16 is a difficult task. Similarly, σ^2 and γ for surrogate gradient of the step function (Eq. 3.16) becomes complex to set in multilayer networks. In this section, we propose a method to estimate some of these parameters based on the network architecture and some assumptions of the probability distribution of the membrane potential.

3.3.1 Analysis of Parameters

The u_{rest} parameter defines the membrane potential (voltage) at which a spike is produced. This parameter can be ignored when its value is equal to zero, according to Equation 2.18. Setting $u_{rest} = 0$ produces a less complex computational graph when the information is propagated in SRNN models. Nevertheless, the contribution of u_{rest} could help to avoid gradient vanishing when $\mathbf{u}[n]$ is close to zero (Eq. 3.18).

$$\frac{\partial \mathbf{u}[n+1]}{\partial \mathbf{h}[n]} = (1-\beta)\left(-\mathbf{u}[n] + u_{rest}\right) + \beta \mathbf{V}. \tag{3.18}$$

We consider the β parameter as a fixed scalar, which is defined as $\beta = \frac{\delta t}{\tau_{mem}}$, and we do not use the δt and τ_{mem} parameters elsewhere. The β parameter plays an important role in SRNN models because it scales the input current $\mathbf{i}^{(l)}[n]$ in the membrane potential $\mathbf{u}^{(l)}[n]$. In addition, scaling $\mathbf{i}^{(l)}[n]$ is necessary to avoid a bad behaviour when a spike occurs. This bad behaviour is produced for large values of $\mathbf{u}^{(l)}[n]$, for instance, when $\mathbf{i}^{(l)}[n] = 10$, $\mathbf{u}^{(l)}[n] = 11$ and $\vartheta = 1$ in Eq. 3.9 almost in every step simulation, the neuron will always produce a spike.

In surrogate gradient, a bad selection of the β parameter produces a bad approximation because $\frac{\partial \Theta(\mathbf{u}[n],\, \vartheta)}{\partial \mathbf{u}[n]}$ is represented by a normal distribution that generates large values of the gradients. For instance, when the SRNN model process a long sequence, a large β could cause exploding gradient, otherwise, a too small β could cause a vanishing gradient. These gradient problems arise because the β parameter is strongly related to the learning rule as can be seen here:

$$\frac{\partial \mathbf{u}[n+1]}{\partial \mathbf{u}[n]} = (1-\beta)\left(1 - \mathbf{h}[n]\right) \tag{3.19}$$

Finally, we analysed the threshold ϑ of the SRNN models based on two cases, having a large and small values of ϑ. On the one hand, a large value of this parameter could produce a slow learning regime because the learnable parameters tend to increase to produce spikes. On the other hand, a small value of ϑ could produce saturated spiking layers because the update of the learnable parameters tends to produce a fast transition between non spikes to spikes events. A solution is to define a small β parameter. In summary, we can conclude that β, u_{rest} and ϑ are strongly related for a good performance of the model.

3.3.2 Analysis of Surrogate Gradient

Surrogate gradient for the step function comprise two factors, σ^2 and γ. As shown in Figure 3.10, the σ factor controls the width or narrowness of the probability distribution where a small variance results in a narrow distribution, and a large variance leads a wide distribution. On the other hand, the γ parameter in Eq 3.16 works as a scaling factor (gradient alleviate value).

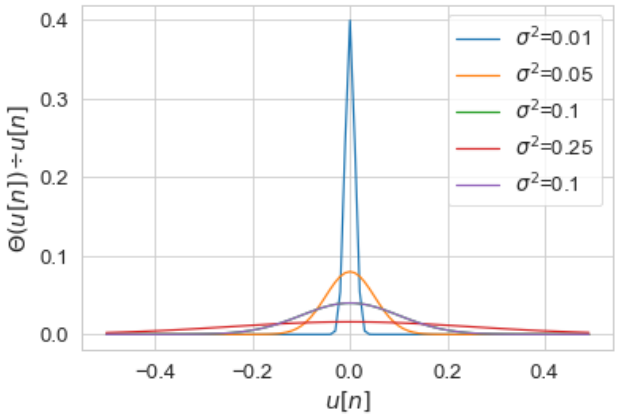


Figure 3.10: Surrogate gradient for different standard deviations for normal distribution

Regarding γ, a large value can cause the gradient to explode, regardless of the σ^2 value. In contrast, a small value can produce stable gradients when σ^2 is also small. When σ^2 is large, the stability of the gradient also depends on $\mathbf{u}[n]$. This happens because $\mathbf{u}[n]$ has a small range for producing errors in surrogate gradient. For instance, as shown in the blue line in Fig. 3.10), if $\sigma^2 = 0.01$ and $\gamma^2 = 0.01$ when $|\mathbf{u}[n]| > 0.02$, the surrogate gradient is zero, but when $|\mathbf{u}[n]|$ is close to zero, the gradients reach their highest point on the graph.

In this work we use mutilayer RNN models for solving both classification tasks where each layer has different number of neurons (Table 3.2). This suggests that σ^2 and γ should be related to the structure of each layer rather than using an literature fixed value for the whole architecture.

3.3.3 Initialisation of Learnable Parameters (Weights)

Heuristic methods are commonly used to initialise the learnable parameters of RNNs. In this work, we use the heuristic initialisation for linear layers proposed in [27], which is defined as follow:

$$a = \sqrt{\frac{2}{n^l}}; \quad \mathcal{U}\left(-a, a\right), \tag{3.20}$$

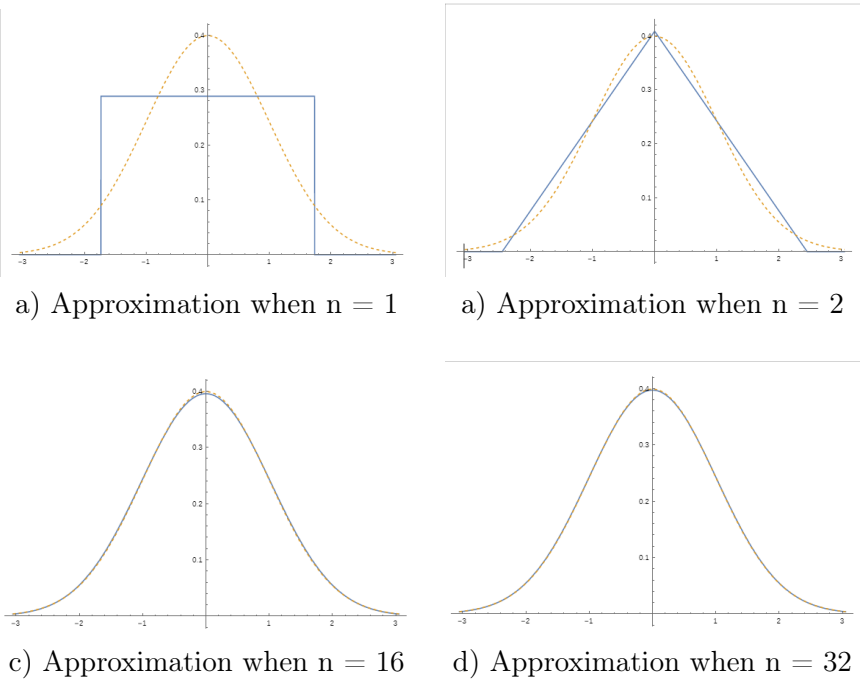
where n^l is the number of neurons in the hidden layer l, and $\mathcal{U}$ represents a continuous uniform distribution with $\mathcal{U}_\mu = \frac{a-a}{2} = 0$ and a standard deviation equal to $\mathcal{U}_\sigma = \frac{2a}{\sqrt{12}}$.

3.3.4 Proposed method to set SRNNs hyperparameters

SRNN formulation comprises parameters such as β, u_{rest} and ϑ, which are present in their dynamics. These parameters play an important role for the correct behaviour in forward

and backward propagation through time because a bad selection could produce a slow or null learning. In the following paragraphs and equations we propose a heuristic method to estimate them automatically.

In Eq. 2.14, $\mathbf{h}^{(l)}[n-1]$ and $\mathbf{h}^{(l-1)}[n]$ are binary vectors, and $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$ are independent uniformly distributed matrices. Based on the **Central Limit Theorem** (CLT) [67] (Figure 3.11), a linear transformation (without bias) between a matrix and a hidden state approximates a value sampled from a normal distribution. For instance, $\mathbf{h}^{(l-1)}[n] \cdot \mathbf{W}$ approximates a normal distribution with $\hat{n}\mathcal{U}_\mu = 0$ and $\sqrt{\hat{n}}\mathcal{U}_\sigma$ standard deviation (Equation 3.21). An empirical simulation of the linear transformation is shown in Appendix A.



a) Approximation when n = 1 a) Approximation when n = 2

c) Approximation when n = 16 d) Approximation when n = 32

Figure 3.11: Illustration of the CLT when we sampling from a continuous uniform distribution on the interval $[-1, 1]$, where orange dashed line is the probability distribution function of the a standard normal distribution and the blue line is the approximation using CLT theorem. Taking from [67]

$$X_{(\mathrm{i})} = \mathcal{N}(\hat{n_\mathrm{i}}\mathcal{U}_\mu = 0, \sqrt{\hat{n_\mathrm{i}}}\mathcal{U}_\sigma), \tag{3.21}$$

where $\hat{n_\mathrm{i}}$ is the activate neurons in the input current. Similarly, $\mathbf{h}^{(l)}[n-1] \cdot \mathbf{V}$ approximates a random variable normally distributed:

$$Y_{(\mathrm{i})} = \mathcal{N}(\hat{n_\mathrm{i}}\mathcal{U}_\mu = 0, \sqrt{\hat{n_h}}\mathcal{U}_\sigma), \tag{3.22}$$

where $\hat{n_h}$ is the activate neurons in the hidden signal.

These linear transformations approximate a normal distribution, so assuming that X and Y are independent random variables, we can obtain the Equation 3.23, where Z represents the input current to the membrane potential defined in the Equation 2.15. Since $\hat{n_\mathrm{i}}$ and $\hat{n_h}$ changes when the model is learning, Z always changes the standard deviation. For example,

when neurons in the input and hidden layers n^l are deactivated, no sampling is effectuated. This case is unlikely because the models need to produce spikes to solve a problem. Another example is when all neurons in the input and hidden layers n^l are activated ensuring a random variable with normal distribution (Demo in Appendix B).

$$Z = X_{(\mathrm{i})} + Y_{(\mathrm{i})} = \mathcal{N}\left(0,\ \mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right) \tag{3.23}$$

In Equation 2.15, Z is scaled with a β factor producing $\mathcal{N}\left(0,\ \beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right)$. Then, βu_{rest} is added up to βZ generating $\mathcal{N}\left(\beta u_{rest},\ \beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right)$. Next, assuming that the previous $\mathbf{u}[n]$ is never reset, Z is scaled again with $(1-\beta)$ factor, which is always aggregated in the membrane potential generating $\mathcal{N}\left(T\beta u_{rest},\ T\beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right)$ in time step n. The above expression is unlikely because spikes are needed to solve the problem.

On the other hand, assuming that there are always spikes, the membrane potential could follow the distribution $\mathcal{N}\left((2(1-\beta)u_{rest},\ \beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right)$ on any time step n. This scenario is also unlikely because the model needs deactivated neurons to solve the problem. Therefore, a mix of the previous scenarios with spikes and non-spikes to compute the parameters are needed. For that, we use the expression $\mathcal{N}\left(\tilde{n}\beta u_{rest},\ \tilde{n}\beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)\right)$, where $\tilde{n}$ could refer to the step when a spike occurs. The previous values are dynamics because when membrane potential evolves in time, the mean and standard deviation also evolve.

With the previous statements, β parameter could be defined by the following rules. First, β must scale the values of $\mathbf{u}[n]$ small enough to give a range of evolution to reach ϑ. Second, this parameter must be related to the architecture of the SRNN network. For instance, a parameter related to the architecture is the a bound from equation 3.20. Finally, β must not be too small, as it could cause problems in learning. Therefore, the candidate to define β parameter is the upper bound a.

Finally, the parameter ϑ should be close to $\tilde{n}$ times $\beta^2\mathcal{U}_\sigma\left(\sqrt{\hat{n_\mathrm{i}}} + \sqrt{\hat{n_h}}\right)$. Therefore, we define ϑ using different values of $\tilde{n} \in \{2, 4, 5, 6, 7, 8, 10, 20\}$. Similarly, u_{rest} should be related to β, so we set $u_{rest} = -\vartheta$. With these steps we reduce three parameters to just one ($\tilde{n}$) deriving to following expression for the threshold:

$$\vartheta(\tilde{n}) = \tilde{n}\left(\frac{1}{\mathrm{d}_h^{(l)}}\frac{2a}{\sqrt{12}}\left(\sqrt{\frac{\mathrm{d}_h^{(l-1)}}{2}} + \sqrt{\frac{\mathrm{d}_h^{(l)}}{2}}\right)\right), \tag{3.24}$$

where $\mathrm{d}_h^{(l-1)}$ is the input dimension signal and $\mathrm{d}^{(l)}$ hidden is the dimension. These values are divided by two because we are assuming that half of these dimensions have spike events. In addition, we are assuming that the standard deviation of learnable parameters changes in small values and its mean is close to zero (considered as a noise value that can be ignored).

Routine Implementation

We implement our proposed heuristic method (Eq. 3.24) using Python 3.7 and PyTorch library, using Object Oriented Programming (OOP) paradigm. This project can be found in the following repository: `https://github.com/jthoth/heuristic-vth-srnn`.

3.4 Memory in SRNNs

Commonly, the memory in SRNNs is analysed following neuroscience principles. Nevertheless, as far as we know, there are no previous works that analyse the memory of SRNNs based on their structure computation which refers to how information flows in the models. Therefore, we categorised the type of memory underlying spiking models, following the framework propose by [45].

To analyse the memory in RNNs, it is necessary to define a new form of building $\mathbf{h}^{(l)}[n]$ expressed as follow:

$$\mathbf{h}^{(l)}[m] = \phi\left(\mathbf{h}^{(l-1)}[m],\ \mathbf{e}^{(l)}[m_2], \mathbf{e}^{(l)}[m_3], \ldots, \mathbf{e}^{(l)}[m_T],\right), \tag{3.25}$$

which decouples the state steps from *memory events*, where, $\mathbf{e}^{(l)}[m]$ represents the relevant information in a layer l and a time step m. Relevant information is a function defined as:

$$\mathbf{e}^{(l)}[m] = f(\mathbf{h}^{(l-1)}[m],\ \mathbf{h}^{(l)}[m-1]), \tag{3.26}$$

where $\mathbf{h}^{(l)}[m-1]$ represents a useful event that happened in past time. Equation 3.26 was formulated because important events do not happen all the time. Then, Equation 2.1 was reformulated as the following two expressions:

$$\mathbf{h}^{(l)}[\mathrm{i}] = \phi\left(\mathbf{h}^{(l-1)}[\mathrm{i}],\ \mathbf{e}^{(l)}[\mathrm{i}-1]\right), \tag{3.27}$$

$$\mathbf{e}^{(l)}[\mathrm{i}-1] = f\left(\mathbf{h}^{(l-1)}[\mathrm{i}],\ \mathbf{h}^{(l)}[\mathrm{i}-1]\right), \tag{3.28}$$

where i represents the time step when an important event occurs.

3.4.1 Categorisation of SRNNs based on their type of memory

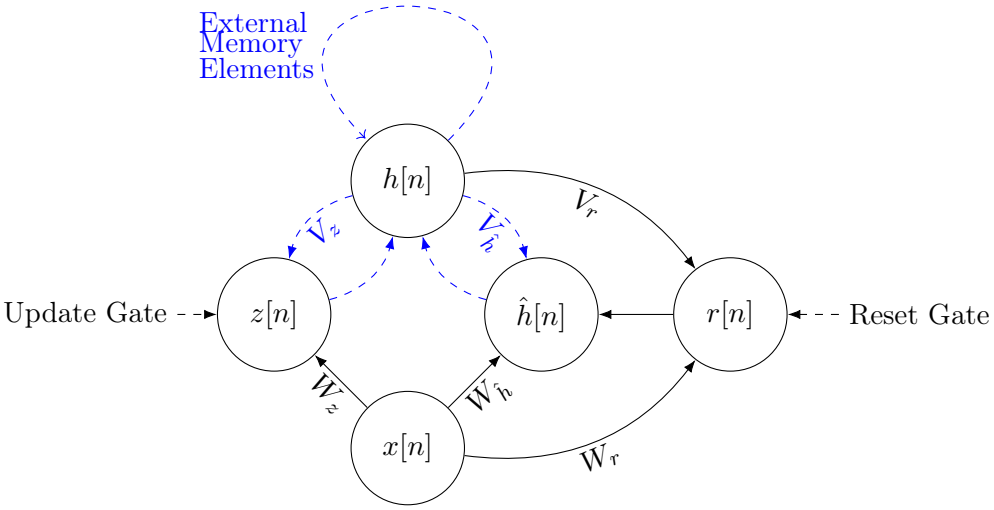
To define the type of memory in SRNNs, we start by describing the known types of memory in RNNs. On the one hand, Equation 2.1, which defines a VRNN, induces memory by mixing all past information into its hidden state. This type of memory, known as **internal memory**, can not recover all the important events. Therefore, the state can also be viewed as a single compound event that is updated at each time step (Figure 3.1) and is expressed as:

$$\mathbf{h}^{(l)}[\mathrm{i}] = \phi\left(\mathbf{e}^{(l)}[\mathrm{i}-1]\right) \tag{3.29}$$

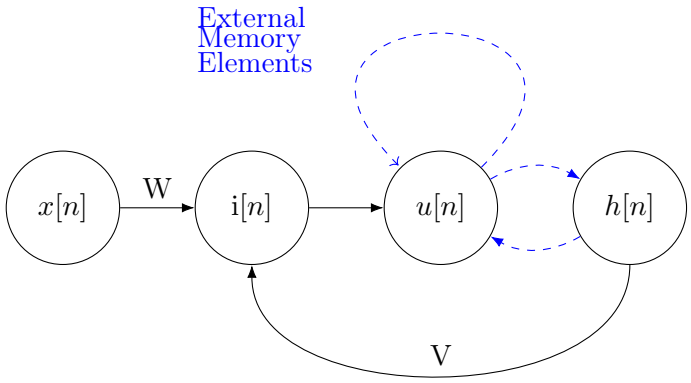
Ma and Principe [45] define an external memory (A mechanism which uses a external element to store information) using LSTM formulation. We follow their methodology using a GRU external memory approach (Equation 3.30) that clearly shows how the external memory arises in this model. This external memory mechanism is produced because there is a controlled feedback connection with previous states as shown in figure 3.12. Finally, GRU external memory can be expressed as $\mathbf{h}^{(l)}[n] = \phi(\mathbf{m}^{(l)}[1])$, where $\mathbf{m}^{(l)}$ indicates that a useful event happened in layer l and time step $n = 1$.

$$\mathbf{m}^{(l)}[n] = (1 - \mathbf{z}^{(l)}[n]) \cdot \mathbf{m}^{(l)}[n-1] + \mathbf{z}^{(l)}[n] \cdot \hat{\mathbf{h}}^{(l)}[n] \tag{3.30}$$

After defining the memory mechanisms for traditional RNNs, we categorised SRNNs as models with **external memory**. This external memory arises due to the connection that exists between the membrane potential with itself and with the hidden states (Figure 3.1). A hidden state works as a gate that controls whether information from the previous membrane potential survives into the next membrane potential state. This external memory mechanism is different from GRU because the gate that decides what information is relevant in GRU is based on a V-RNN and in SRNN it is based on its dynamics without learnable parameters (Fig. 3.12).



a) External memory elements in GRU graph



b) External memory elements in V-SRNN graph

Figure 3.12: Illustration of the external memory elements in GRU and V-SRNN models

3.4.2 Adaptation of Two-Level External Memory in SRNNs

In the previous section, we categorise the type of SRNNs memory in internal and external. In order to add external memory to V-SRNN, we used a GRU based formulation. The external

memory in the hidden state of a V-SRNN is based on replacing the Equations 2.7, 2.8, 2.9 with V-SRNNs and maintaining the definition of the hidden state of GRU (Equation 2.10).

$$\mathbf{r}^{(l)}[n] = \text{V-SRNN}_{\mathbf{r}}\left(\mathbf{h}^{(l)}[n-1],\ \mathbf{u}_{\mathbf{r}}^{(l)}[n],\ \mathbf{h}^{(l-1)}[n]\right) \tag{3.31}$$

$$\hat{\mathbf{h}}^{(l)}[n] = \text{V-SRNN}_{\hat{\mathbf{h}}}\left(\mathbf{r}^{(l)}[n] \odot \mathbf{h}^{(l)}[n-1],\ \mathbf{u}_{\hat{\mathbf{h}}}^{(l)}[n],\ \mathbf{h}^{(l-1)}[n]\right) \tag{3.32}$$

$$\mathbf{z}^{(l)}[n] = \text{V-SRNN}_{\mathbf{z}}\left(\mathbf{h}^{(l)}[n-1],\ \mathbf{u}_{\mathbf{z}}^{(l)}[n],\ \mathbf{h}^{(l-1)}[n]\right) \tag{3.33}$$

For formulation purposes, we define $R_t = \mathbf{r}^{(l)}[n]$, $\hat{H}_t = \hat{\mathbf{h}}^{(l)}[n]$ and $U_t = \mathbf{z}^{(l)}[n]$. Since R_t, $\hat{H}_t$ and U_t states are binary sets ($R_t, \hat{H}_t, U_t, \in S$), a binary operation closed under S (a non-empty binary set) is required to compute the hidden state H_t. The hidden state in the GRU model is a linear interpolation. This interpolation on the set of binary numbers is a closed operation. Therefore, the binary operation $(1-Z_t)\hat{H}_t + Z_t H_{t-1}$ on S is a closed binary operation on S if $(1 - Z_t)\hat{H}_t + Z_t H_{t-1}$, $\forall\ U_t,\ \hat{H}_t\ \in S$.

We call to the previous formulation Gated SRNN (**G-SRNN**), which has an external memory at two levels (membrane potential and hidden state). It can be seen as GRU model with binary gates and hidden states (Fig. 3.12). In addition, it has the property of alleviating the vanishing gradient problem because it inherits the GRU's ability to automatically implement the Truncated Backpropagation Through Time (**TBPTT**).

Routine Implementation

We code our proposed G-SRNN (Eq. 3.31, 3.32, 3.33) using Python 3.7 and PyTorch library, using Object Oriented Programming (OOP) paradigm. This project can be found in the following repository: `https://github.com/jthoth/gated-srnn`.

3.5 Information Bottleneck in SRNNs

Similar to ANNs, the SRNN can be characterised as a Markov chain in space (number of layers) but also in time. However, characterising in time adds computational complexity to calculate the information measures. For that reason, we decided to characterise SRNN model as a Markov chain in space, ignoring the time component. Space Markov chain in SRNNs can be expressed as:

$$\underbrace{H_t^{(0)}}_{\mathrm{X_t}} \rightarrow \underbrace{H_t^{(1)} \rightarrow H_t^{(2)} \rightarrow \cdots \rightarrow H_t^{(L-1)}}_{\hat{\mathrm{X}}_t} \rightarrow \underbrace{H_t^{(L)}}_{\hat{\mathrm{Y}}_t}, \tag{3.34}$$

where $H_t^{(0)}$ represents the input signal, $H_t^{(1)}$ represents the first hidden signal and $H_t^{(L)}$ represents the output signal (encoder linear mapping). All hidden layers could represent a whole hidden state signal, which can be expressed as $\hat{\mathrm{X}}_t$. These signals are binary random variables composed of the number of samples and neurons.

In order to incorporate the space deterministic Information Bottleneck in V-SRNN (with the best architecture and model parameters), we propose the Equation 3.35.

$$\mathcal{L}_{IB} = \mathrm{CE}\left(Y, \frac{\sum_t \hat{\mathrm{Y}}_t}{T}\right) - \gamma \frac{\sum_t I\left(X_t;\ \hat{X}_t\right)}{T}, \tag{3.35}$$

where CE is the cross entropy function, I is the mutual information and γ is the Lagrange parameter. This formulation is represented in the Figure 3.13.

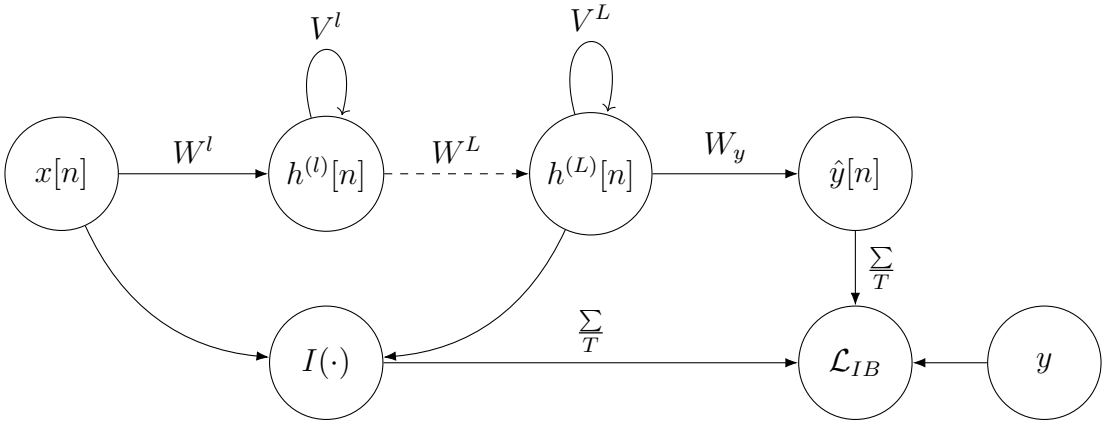


Figure 3.13: Information Bottleneck in V-SRNN.

To compare the performance of the IB method with respect to the cross entropy, we selected the best architecture among the evaluated SRNNs. For that, we consider the number of spikes generated per layer and the accuracy metric. In addition, we evaluate several mutual information estimators such as those proposed by Sanchez et al. [55], Kolchinsky and Tracey [36], and an MI estimator proposed in this work based on the *Central Limit Theorem*. This comparison is based on the evaluation of different values of Lagrange parameter (varying γ).

3.5.1 Proposed Mutual Information Estimator Based on the Central Limit Theorem

Since we are dealing with binary random variables, we can assume that they follow a Bernoulli distribution. Based on the above assumption, the probability of a spike can be found as $\mathbb{E}(X)$ through the maximum likelihood estimation. Then, the whole random variable can be reduced adding up all samples in order to obtain a normal distribution based on the central limit theorem $S(X_t) = \mathcal{N}\left(N\mathbb{E}(X_t), \sigma(X_t)\right)$.

$$\sigma(X_t) = \sqrt{N}\sqrt{(\mathbb{E}(X_t)(1 - \mathbb{E}(X_t)))}, \tag{3.36}$$

where N is the number of samples and $p = \mathbb{E}(X_t)$ is the spike probability.

Function $S(X_t)$ allows us to compute the differential mutual information because it defines a valid probability distribution. This computation is possible because the differential mutual information between two normal distributions are well-defined. This principle is used to compute the mutual information for the Information Bottleneck method. In addition to calculating the mutual information, the proposed estimator's characteristic is that its operations are differentiable.

$$I(X_t; Y_t) = \frac{1}{2} log\left(1 + \frac{\sigma(X_t)}{\sigma(Y_t)}\right) \tag{3.37}$$

Routine Implementation

We implement our estimator (Eq. 3.37) using Python 3.7 and PyTorch library, using Object Oriented Programming (OOP) paradigm. This class can be found in the following repository: `https://github.com/jthoth/guassian-clt-mi`.

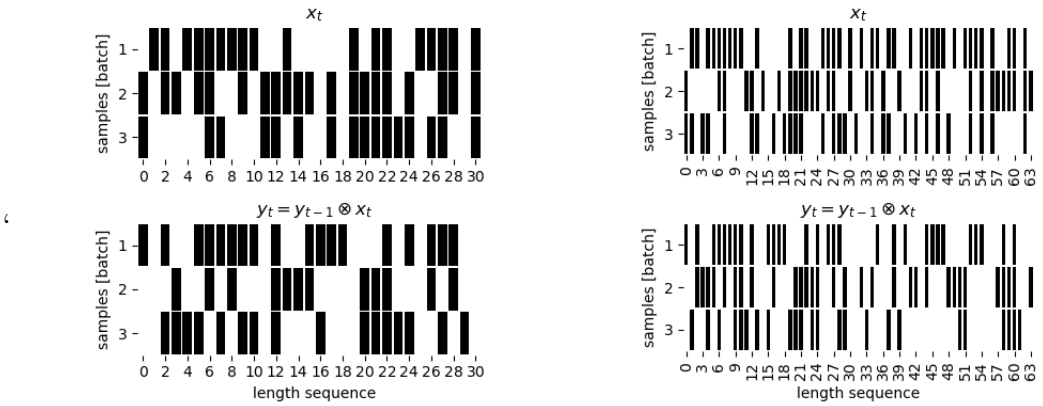
Chapter 4

Baseline Results

The aim of this chapter is to create a proof of concept and a baseline for spiking models. The baseline allows making a fair comparison between the spiking models and traditional recurrent networks. All spiking and non-spiking models are trained with the methods described in the Section 3.2. Our proposed training and evaluation framework comprises the construction of the baseline using two datasets with different architectures and lengths for training the models. For the evaluation, we report the best accuracy obtained in the test set. To analyse our results, we create heat map graphs describing the accuracy obtained by each training setup.

4.1 Parity Bit Problem

We use the Parity Bit dataset in the proof of concept to contrast theory with practice for the models studied. This dataset is used exclusively for this purpose. In total, we created 4 different versions of this dataset by varying its length (see Table 3.1). Figure 4.1 shows samples generated using 32 and 64 time steps.



a) Samples generated with 32 time steps b) Samples generated with 64 time steps

Figure 4.1: Parity Bit dataset visualisation with different time length versions, where x_t represent the input and y_t the output sequence.

The heat map shown in Figure 4.2 outlines the maximum accuracy (effectiveness) achieved on the test dataset by different models. The x-axis indicates spiking and non-spiking models with their respective architectures (y-axis). Overall, all models solved the problem for sequence lengths of 16 and 32. Nevertheless, V-RNN and A-SRNN tended to fail when the sequence becomes larger.

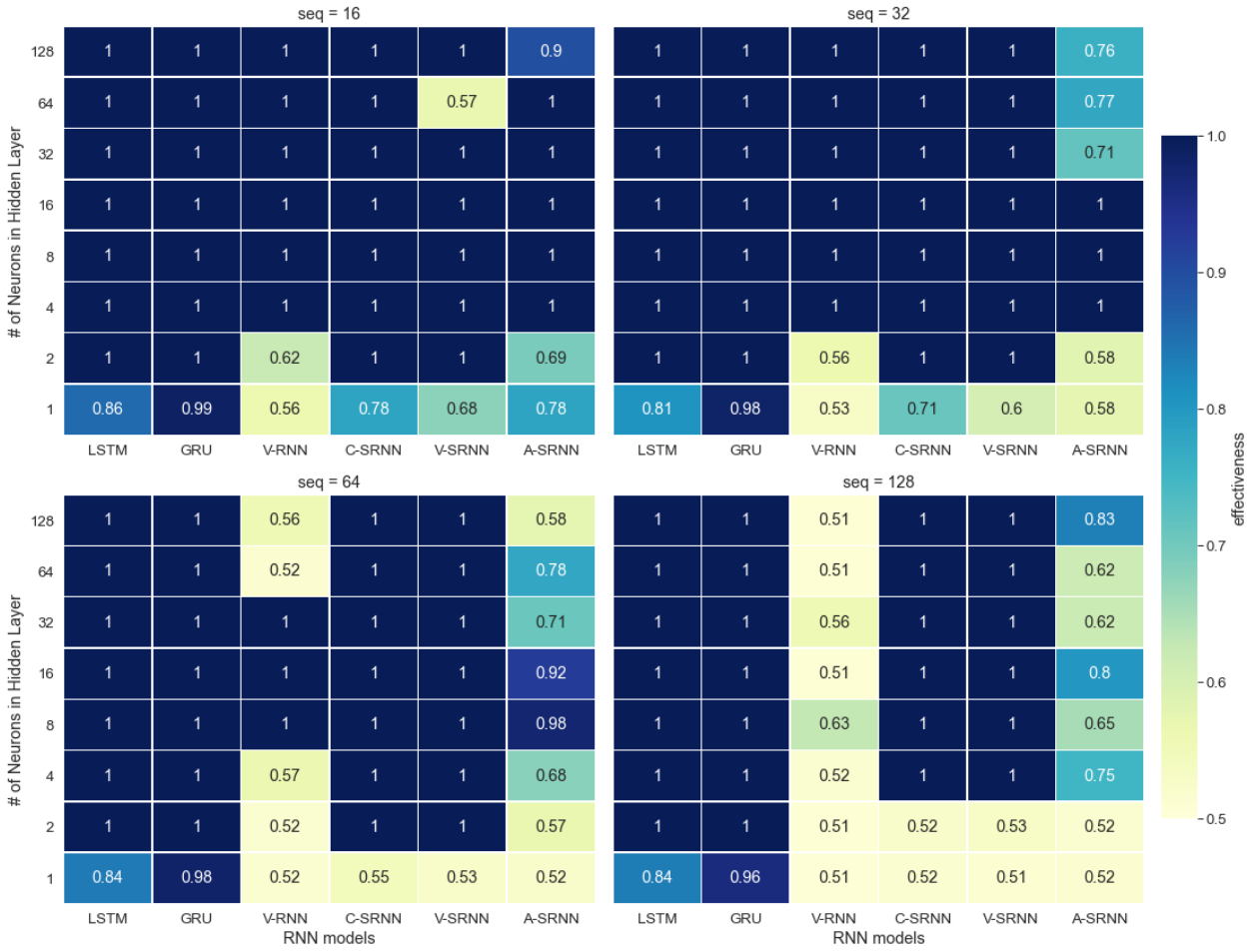


Figure 4.2: Parity Bit test accuracy heat-maps by different types of cells, number of neurons and sequence lengths.

For a sequence length equal to 16 (top left), all models, except the V-RNN, achieved a good performance. However, no model with a single neuron was able to solve this problem. Using 2 neurons, V-RNN and A-SRNN failed to solve this problem when the models reach 10 training epochs.

For a sequence length equal to 32 (top right), all models, except the A-RNN, achieved a good performance when the model reached the maximum number of epochs of training. In this scenario, LSTM, GRU, C-SRNN and V-SRNN models reached the maximum score in most architectures, except when using only a single recurrent neuron in the hidden layer. On the other hand, A-SRNN had problems finding a solution with the number of epochs assigned.

The sequence length equal to 64 reveals the limitations of V-SRNN and A-SRNN models.

In the case of V-SRNN, few architectures were able to solve this problem with the previous analysed sequences. For A-SRNN model, no architecture managed to converge to the maximum accuracy. Similar to the shorter sequences, the single-neuron architecture did not solve the problem, with the exception of LSTM and GRU.

Finally, for a sequence length equal to 128, none of the V-RNN architectures solved the problem. Similarly, in the case of A-SRNN, no architecture achieved the maximum accuracy, however this model performed better than V-RNN. For C-SRNN and V-SRNN, unlike the other scenarios, the architecture with 2 neurons in the hidden layer did not solve the problem. According to our results, LSTM, GRU, C-SRNN and V-SRNN models achieved the best performance solving the parity bit problem with a least 4 neurons in the hidden layer. Notice that the SRNN models have less parameters compared with LSTM and GRU. The scatter plots shown in Figure 4.3 help to analyse the results of the different sequence lengths by type of cell (colour and shape), number of learnable parameters (x-axis), accuracy (y-axis) and number of neurons in the hidden layer (size).

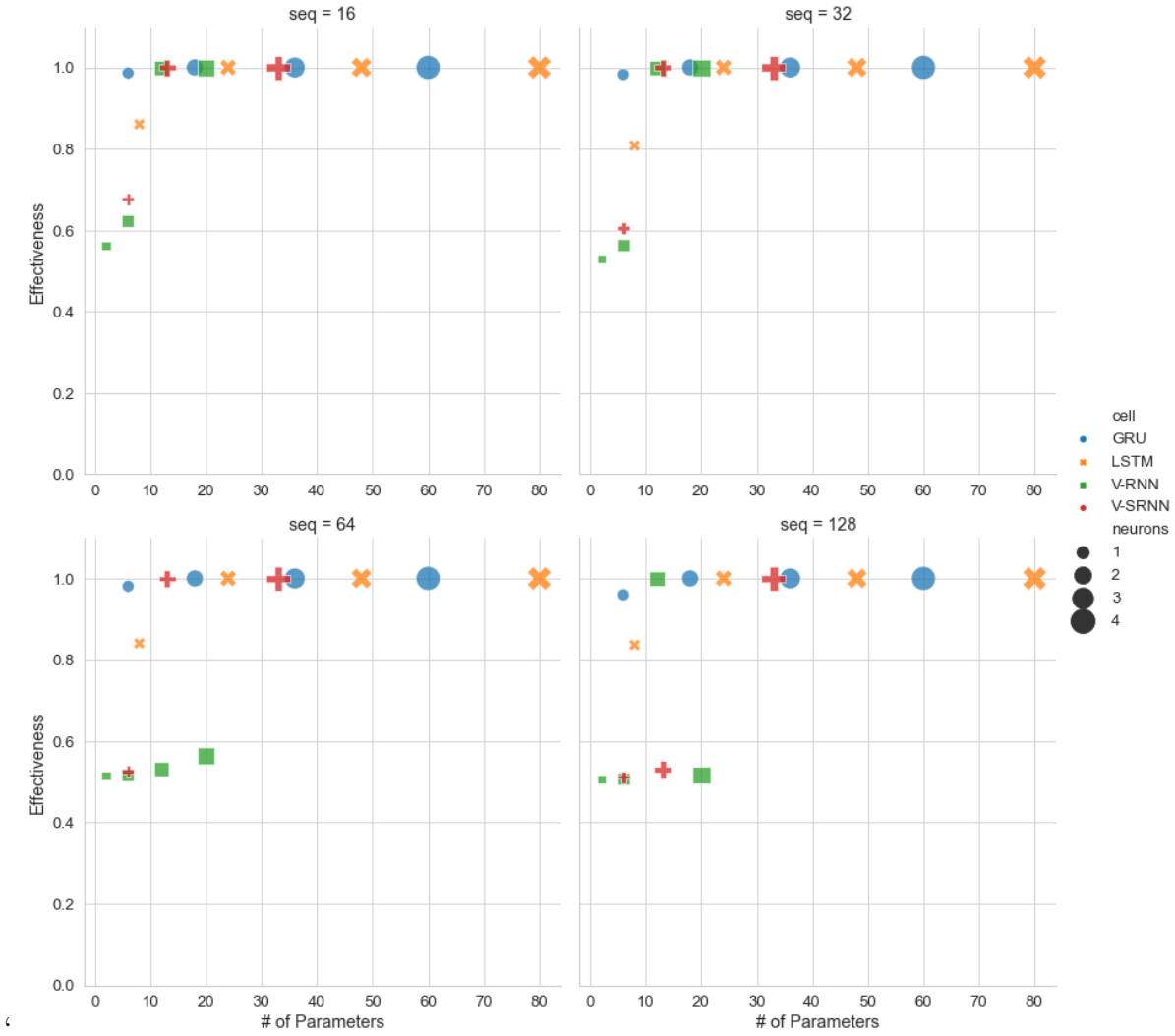


Figure 4.3: Scatter plot of parity bit test accuracy versus number of cells, neurons, parameters and sequence length.

The scatter plot shows the differences between the models and architectures with respect to their number of parameters. For instance, the LSTM and GRU models with 4 neurons in the hidden layer have more parameters than the other models for all sequences. For sequences of length 16 and 32, all models reached the max performance with at least 10 parameters. In contrast, for a sequence of 64, V-RNN did not solve the problem, while for a sequence length of 128, it solved the problem with 3 neurons.

4.1.1 Discussion

As mentioned in the literature review, RNN models are Turing Complete because they can simulate any arbitrary program with proper weights. Since SRNNs are a family of RNNs, by transitivity they are Turing complete. This fact suggests that spiking models can simulate the program underlying the parity bit generator function (Equation 3.2).

The current study found that almost all models can solve parity bit problem for different sequence lengths. Nevertheless, V-RNN tends to have some issues for larger sequence lengths. A possible explanation for these results may be the lack of adequate control of the vanishing gradient problem, which is solved in LSTM and GRU models, and partially solved in spiking models.

These findings further support the notion that LSTM and GRU cells work well with long sequences. However, the number of parameters of these models is much higher than those of spiking and V-RNN models. Surprisingly, LSTM, GRU and V-SRNN models achieve competitive performance, regardless of the number of neurons in the hidden layers.

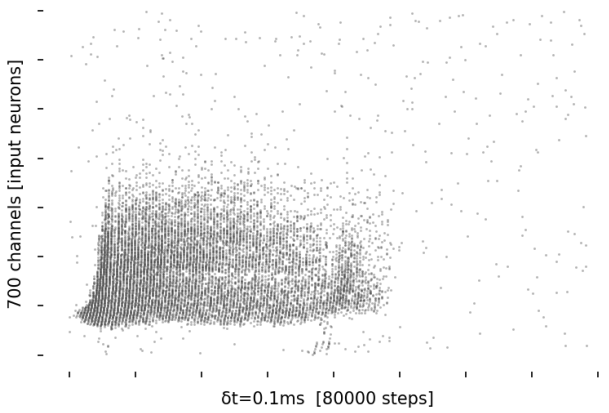
In response to our first research question **RQ1**: *What is the minimum number of parameters needed to solve the classification tasks used in this work, regardless of the RNN or SRNN architecture?*, our findings suggest that almost all models can solve the parity bit problem using at least 10 parameters, regardless of the number of neurons in their hidden layers.

4.2 SHD Problem Baselines

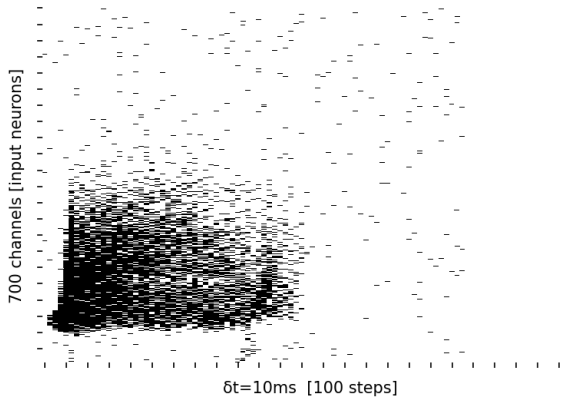
To classify the real-world Spiking Heidelberg dataset, we generated a baseline for each training configuration, which comprises the type of model, architecture, training strategies, etc, following the pipeline detailed in Figure 3.9. Previous works studied the LSTM, C-SRNN and A-SRNN models for specific architectures. However, for the rest of the architectures and models proposed in this work, as far as we know, there are no baselines. With respect to the SRNN models, our baselines are based on the configuration parameters found in [14], [70].

As for the C-SRNN performance, in [14] it was reported 71% accuracy as a baseline, and 83% accuracy after applying multiple data processing techniques and increasing the number of neurons. In addition, the same authors [14] reported the performance of LSTM model using different time step sizes. In Figure 4.4, we show the generated versions of SHD dataset, when using 4ms and 10ms simulation step sizes.

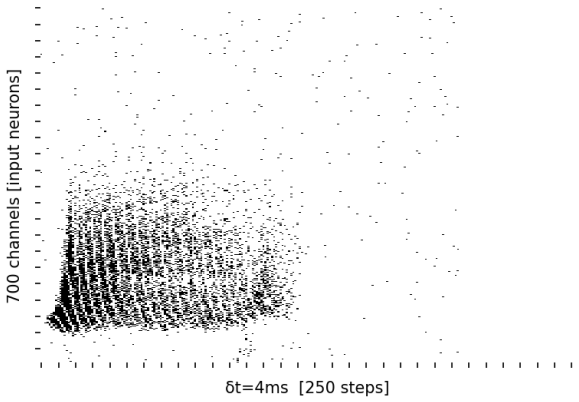
The heat map chart in Figure 4.5 shows the average accuracy (effectiveness) along with the standard deviation on the test set achieved by the spiking and non-spiking models (x-



Sample from the original dataset



a) Sample generated with 10ms time steps simulation (100 size steps)



b) Sample generated with 4ms time steps simulation (250 size steps)

Figure 4.4: Spiking Heidelberg dataset sample visualisation with different time length versions.

axis) with their respective architectures (y-axis). All models, with the exception of V-RNN (for a sequence length equal to 250), solved the SHD problem in at least one architecture with equal or higher accuracy than the C-SRNN reported in [14].

Using 100 steps of sequence length, almost all models have a good performance. Even V-RNN was able to solve this task with good performances in some architectures. The architectures for SRNNs obtained an accuracy from 70% to 77%. In contrast, LSTM and GRU models achieved an accuracy between 81% and 89%. On average, the best model was LSTM with 512 units in its hidden layer (architecture 700-512-20).

For a sequence length equal to 250 steps, V-RNN obtained the worst performance followed by the spiking models (V-SRNN, A-SRNN and C-SRNN) with an accuracy from 39% to 78%, which decreased as more layers were added. With respect to LSTM and GRU models, they achieved an accuracy from 79% to 89%.

Figure 4.5: Mean accuracy and standard deviation of traditional RNNs and SRNN models for the SHD dataset.

4.2.1 Discussion

The current study found that SRNN models performance drops down when more layers are stacked. A similar phenomenon occurs when the sequence length increases. A possible explanation for this might be the gradients or the definition of the parameters. An unexpected finding is that the performance of A-SRNN with three layers obtained the worst accuracy between all models, which might be related to the aforementioned possible explanation.

For C-SRNN benchmark, we increase the results reported in [14] by 5% accuracy using the 700-128-20 architecture. The authors [14] obtained 71.4% accuracy and we achieved 77% accuracy using our proposed training framework. This result may be explained by the fact that we used RNN training techniques to control the exploding gradient and the learning rate. These results support the use of these techniques in the next experiments.

Our V-RNN obtained an average accuracy of 81% solving the SHD dataset. Surprisingly, our V-RNN achieved a maximum accuracy of 83% (as shown in Appendix C.1) which is competitive with the A-SRNN score reported in [70], and also with our small network architecture of GRU and LSTM. This finding was unexpected and suggests that it might be caused by another variable not considered in our analysis. In the following chapters, we extend the analysis of the models not only in terms of accuracy, but also in terms of number

of parameters and iterations in converging to the maximum score, outlining the differences and similarities between spiking and non-spiking models. In addition, we analysed other aspects in SRNN models such as the parameters definition, memory and space compression representation.

Chapter 5

SRNN Model Hyperparameters

In this chapter, we summarise the best results obtained by applying our proposed heuristic estimation of the SRNN model hyperparameters (Section 3.3). In addition, we compare these results with the performance obtained by the spiking and non-spiking models reported in the previous section. For that, we visualise the results using heatmap and parallelogram chartss.

In summary, we first explore the impact of the $\tilde{n}$ parameter in our proposed method, using the V-SRNN model. Second, we compare the spiking and non-spiking models in terms of accuracy, number of parameters and convergence steps. Finally, we discuss the capabilities and weaknesses of SRNN models. This chapter provides insight into the best SRNN model and architecture for solving the SHD problem based on heuristic estimation of the parameters.

5.1 Heuristic Method Exploration

As mentioned in the methodology section, we explored the impact of $\tilde{n}$ in our proposed heuristic method, which is the average value when a spike occurs. We evaluated the performance of the V-SRNN by varying the value of this parameter, using the following values: $[2, 3, 4, 5, 7, 8, 10, 20]$. The evaluation comprises the average validation accuracy of the model with different architectures and versions of the SHD dataset (i.e., with different time length versions). The heat map chart shown in Figure 5.1 outlines the average accuracy (effectiveness) along their respective standard deviations reached with different architectures (y-axis) and $\tilde{n}$ values (x-axis). In general, a good performance was achieved for almost all values of $\tilde{n}$ in the evaluated architectures and the different sequence lengths.

For the SHD problem with 100 steps of length sequence, few values of $\tilde{n}$ had a bad performance. For the most complex architecture (with more stacked layers), the performance decreased in the extreme values of the evaluation $[2, 20]$. In contrast, the center values of the grid obtained the best scores, for example, when $\tilde{n}$ ranges from 3 to 8 in this architecture of three stacked V-SRNN layers, the model reached an average accuracy of 83%. For the SHD problem with 250 steps of length sequence, it can be noted a similar phenomenon to the case of 100-step dataset, but with lower performances, achieving an accuracy of 40% and 64% with a $\tilde{n}$ equal to 2 and 20, respectively.

Overall, the model scores studied were more stable than the baseline results in both versions of the dataset (Figure 4.5). Furthermore, the most surprising aspect of our results was that with $\tilde{n}$ equals 4, the V-SRNN performed well regardless of architecture and sequence size. In addition, the performance of single-layer models was stable for almost all $\tilde{n}$ evaluation values.

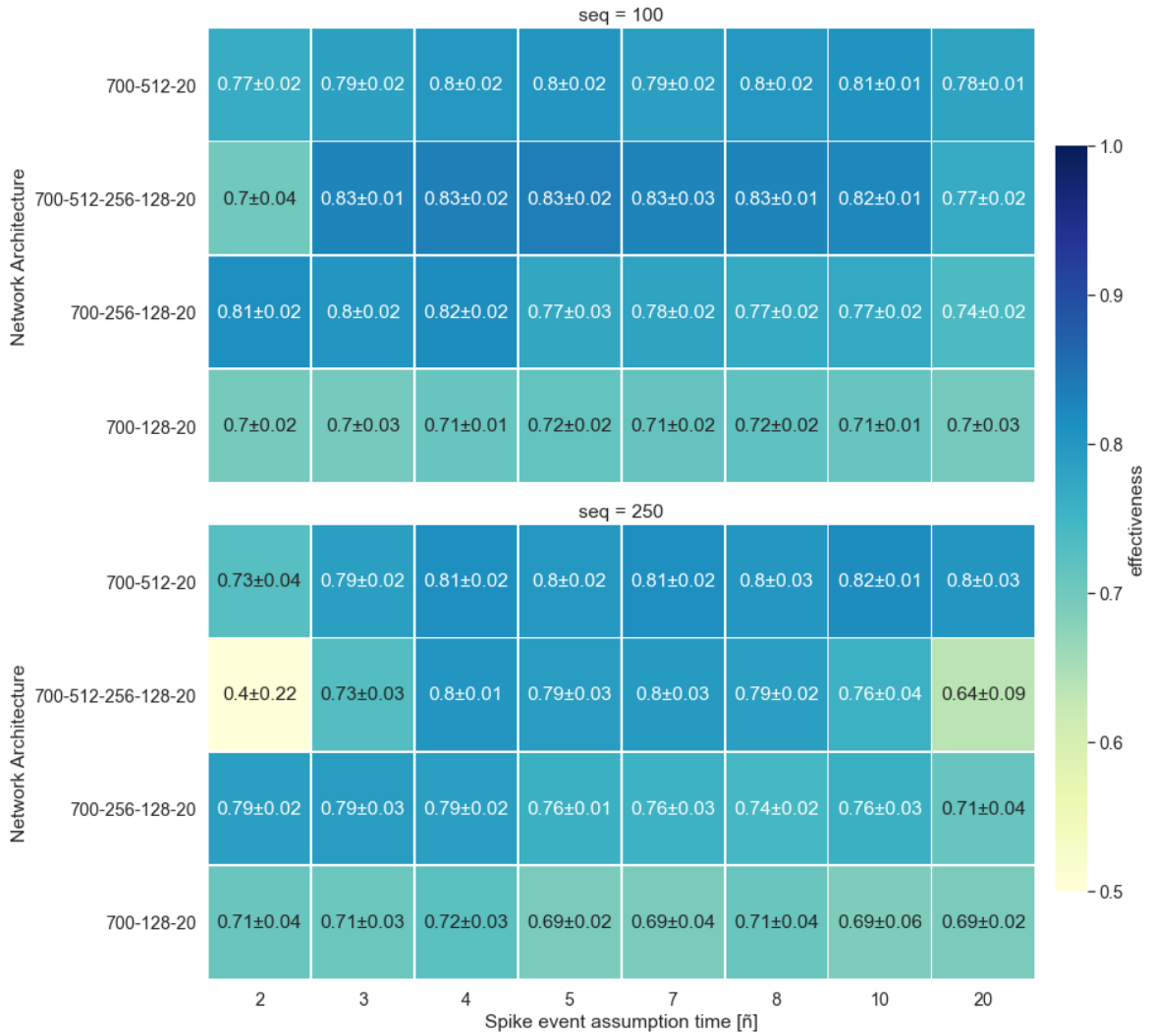
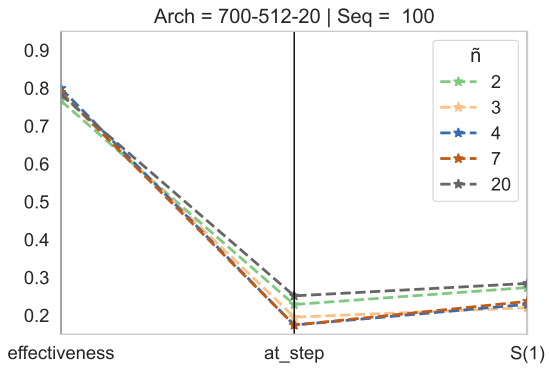
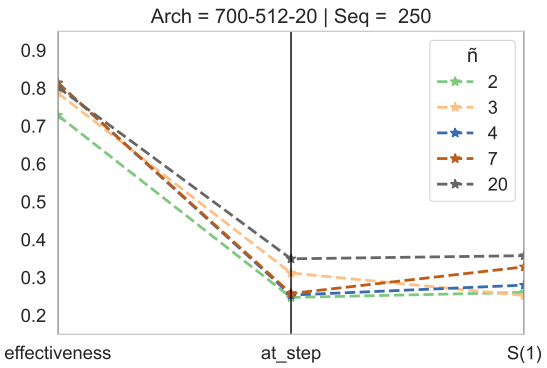


Figure 5.1: Mean accuracy and standard deviation of V-SRNN with different values of the parameter $\tilde{n}$ in the classification of the SHD test data set.

The parallelogram chart shown in Figure 5.2 compares the average accuracy (effectiveness), the convergence training steps and the average number of spikes when evaluating a V-SRNN with five values of $\tilde{n}$ (represented by the line colour) using the *700-512-20* architecture. The at_step variable represents the normalised number of iterations (3k iterations). The $S(1)$ is the average number of spikes occurring in space and time (batch and sequence) in a specific layer. Overall, when the threshold value is large, the model on average tends to converge slowly compared with the best $\tilde{n}$ value. In both dataset versions, for $\tilde{n}$ equal to 4, the V-SRNN converged faster and obtained a better effectiveness than using other values for this parameter. In terms of the number of spikes, the best efficiency was obtained with few spikes per layer. However, the dataset versions behaved differently.



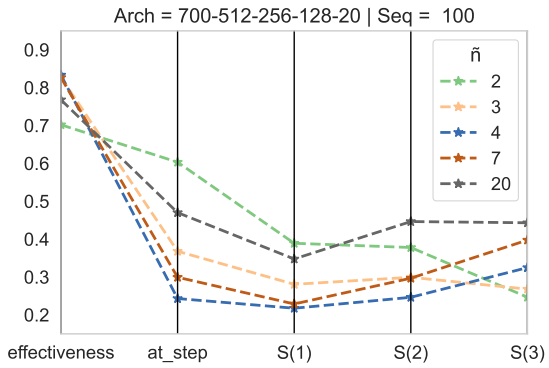
a) 10ms step simulation (100 size steps)



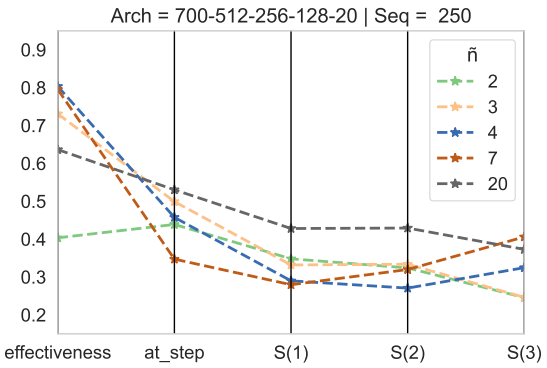
b) 4ms step simulation (250 size steps)

Figure 5.2: Parallelogram chart with the effectiveness, convergence steps, and number of spikes of the V-SRNN using the 700-512-20 architecture.

The parallelogram chart shown in Figure 5.3 presents an analysis of the largest architecture studied in this work (*700-512-256-128-20* architecture). Using the 10ms simulation step (left chart), with a small $\tilde{n}$, the model obtained a low effectiveness, learned slowly and its first layers almost saturated, while with $\tilde{n}$ equal to 4 the convergence of the model improved using fewer spikes per layer. Using the 4ms simulation step (right chart), the model obtained on average the best convergence and good effectiveness with $\tilde{n}$ equal to 7, while with $\tilde{n}$ equal to 20, the model tended to learn slowly and to saturate its spike layers, resulting in the worst effectiveness. In general, some of the $\tilde{n}$ values obtained a similar effectiveness to each other, however, there was one configuration where this metric was also related to a good convergence step and few spikes per layer.



a) 10ms step simulation (100 size steps)

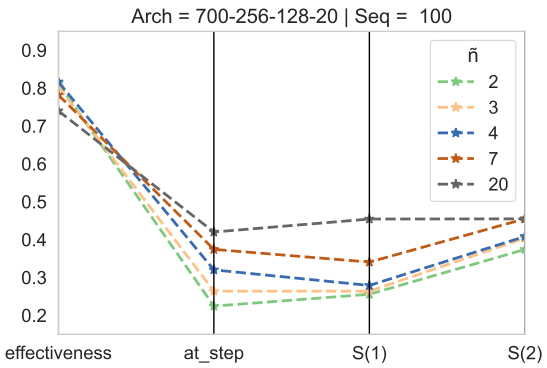


b) 4ms step simulation (250 size steps)

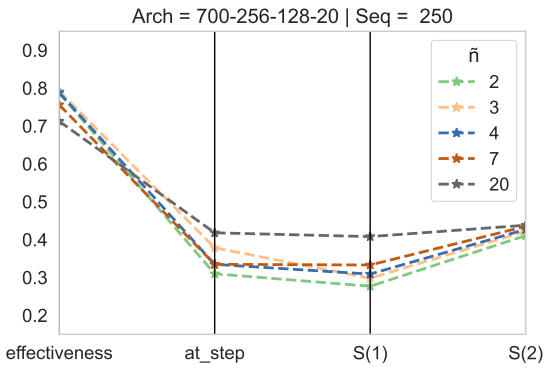
Figure 5.3: Parallelogram chart with the effectiveness, convergence steps, and number of spikes of the V-SRNN using the 700-512-256-128-20 architecture.

The parallelogram chart shown in Figure 5.4 shows the performance of the V-SRNN with two layers (*700-256-128-20* architecture). Overall, the majority of the $\tilde{n}$ values in both dataset versions obtained similar effectiveness, except for $\tilde{n} = 20$. Nevertheless, the convergence steps and spikes per layer changed radically depending on the length of the sequence.

Finally, Figure 5.5 shows the results of the V-SRNN using the *700-128-20* architecture,



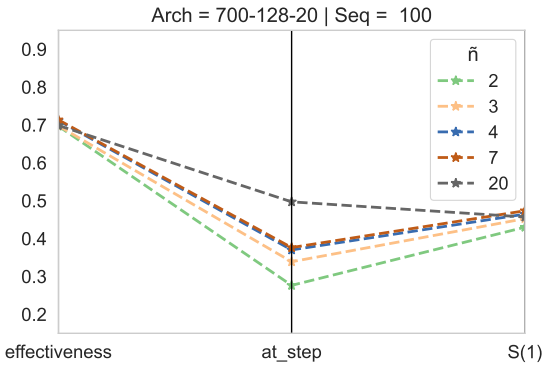
a) 10ms step simulation (100 size steps)



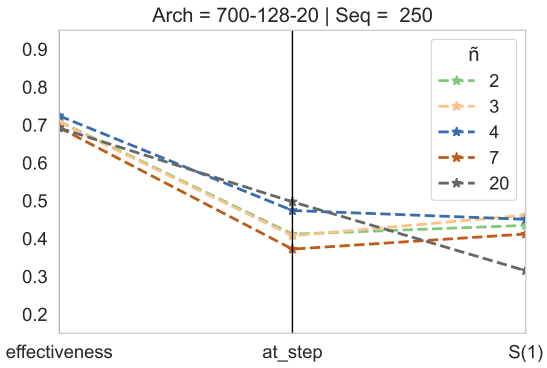
b) 4ms step simulation (250 size steps)

Figure 5.4: Parallelogram chart with the effectiveness, convergence steps, and number of spikes of the V-SRNN using the 700-256-128-20 architecture.

where $\tilde{n} = 20$ obtained bad results and $\tilde{n} = 4$ preserved its best results as in the previously analysed single-layer architecture. Interestingly, for this architecture the performance was invariant to the dataset length version. In addition, another interesting behaviour occurs with the 100-step sequence when $\tilde{n} = 2$ because the model converged faster than the other configurations and also had fewer spikes in its hidden layer.



a) 10ms step simulation (100 size steps)



b) 4ms step simulation (250 size steps)

Figure 5.5: Parallelogram chart with the effectiveness, convergence steps, and number of spikes of the V-SRNN using the 700-128-20 architecture.

5.1.1 Discussion

The present study was designed to determine the effect of $\tilde{n}$ in different architectures of V-SRNN. The findings suggest that there is a range of $\tilde{n}$ values where more spike events are triggered. In turn, these events stabilise the performance of the model, maintaining a balance between the results reported in both dataset versions.

Prior studies that have noted the importance of SRNN models parameters, usually adjust these parameters to a specific architecture. This form of adjusting has an arguable weakness due to the arbitrary nature of its definition. The present study provides a set of steps and

a heuristic method to define the parameters. Overall, one of the most significant findings of this study is that a good setting of parameters generates effective and efficient SRNN models.

The lowest results reported by 3 layers V-SRNN (Figure 5.3) might be related to the small value of $\tilde{n}$ (when it is equal to 2) which generates small threshold ϑ and u_{rest}. Such values could generate a rapid transition between neurons that are activated and non-activated and vice versa. This transition might lead to a limitation in the computational power of SRNNs, as the number of solutions is limited by the threshold and the u_{rest} value.

On the other hand, a large $\tilde{n}$ (when it is equal to 20) generates large values for the threshold ϑ and u_{rest}, so the model has to change the learnable parameters in order to cross this threshold and then reset the membrane potential $\mathbf{u}^{(l)}[n]$. This may cause problems related to model convergence and gradients.

Regarding the first phenomenon, the learning algorithm might need more steps to cross a large threshold, leading to slow learning. This was noticeable in almost all the architectures trained for both versions of the datasets (Figure 5.3). The second phenomenon is caused by the surrogate gradient of the step function being modelled with a normal distribution with zero mean. When the threshold is large, the membrane potential tends to be large which impairs learning as these values move away from the average of the normal distribution (Figure 3.10), producing no learning.

In response to our second research question **RQ2**: *Can we automatically estimate the underlying hyperparameters of SRNNs based on their architecture?*, our findings suggest that we can automatically estimate the hyperparameters of SRNNs based on their architecture for the SHD problem. In addition, there is an optimal range of values for these parameters, nevertheless, some values are more efficient than the others.

The exposed results are on average better than those reported by [14] and [70]. The best parameters obtained with the V-SRNN model are evaluated with other traditional SRNNs and RNNs in the following section.

5.2 SRNNs vs Traditional RNNs

The aim of this section is to compare the performance of the SRNN models with respect to traditional RNNs in terms of effectiveness, convergence steps, spikes and number of parameters. The SRNN models were trained with the heuristic method to compute their threshold ϑ and u_{rest} parameters. We evaluated all models in the classification of the SHD dataset, following the proposed experimental framework. The heat map shown in Figure 5.6 outlines the average accuracy (effectiveness) with their respective standard deviation reached by the models (x-axis) with different architectures (y-axis). Overall, spiking models with 3 layered SRNN obtained similar performances to those of some architectures of LSTM and GRU models.

Figure 5.6: Mean accuracy and standard deviation of SRNNs with heuristic function and traditional RNNs in the classification of the SHD test data set.

For the 100-step sequence, all spiking models obtained superior effectiveness than V-RNN and some architectures achieved competitive results compared to gated RNNs. In addition, spiking models with multiple layers performed better than others spiking architectures. For the 250-step sequence, some architectures decreased their performance compared to the 100-step version, but the single hidden layer architectures maintained a similar performance. Since the gated models have a larger number of parameters compared to the spiking models,

a different analysis was performed to make a fair comparison as reported below.

The scatter plot in Figure 5.7 presents a comparative analysis between the effectiveness (y-axis) and the number of parameters (x-axis) of the spiking and non-spiking models. Overall, spiking models obtained significant improvements using the heuristic method and in some cases, the V-SRNN model obtained better average score than the other models (spiking and non spiking).

Focusing on the 100-step sequence version, all spiking models tended to increase in effectiveness as more parameters were added. Analysing the models with less than 1M parameters, none of them reached the max effectiveness reported. Nevertheless, some of these models perform similarly to each other, for instance V-SRNN, GRU and LSTM. On the other hand, when sequence has 250 steps, not only V-SRNN is affected in its performance but also most other models. However, the performance of spiking models are close to the performance of the gated models. For V-RNN, the performance is severely affected despite the use of gradient clipping.

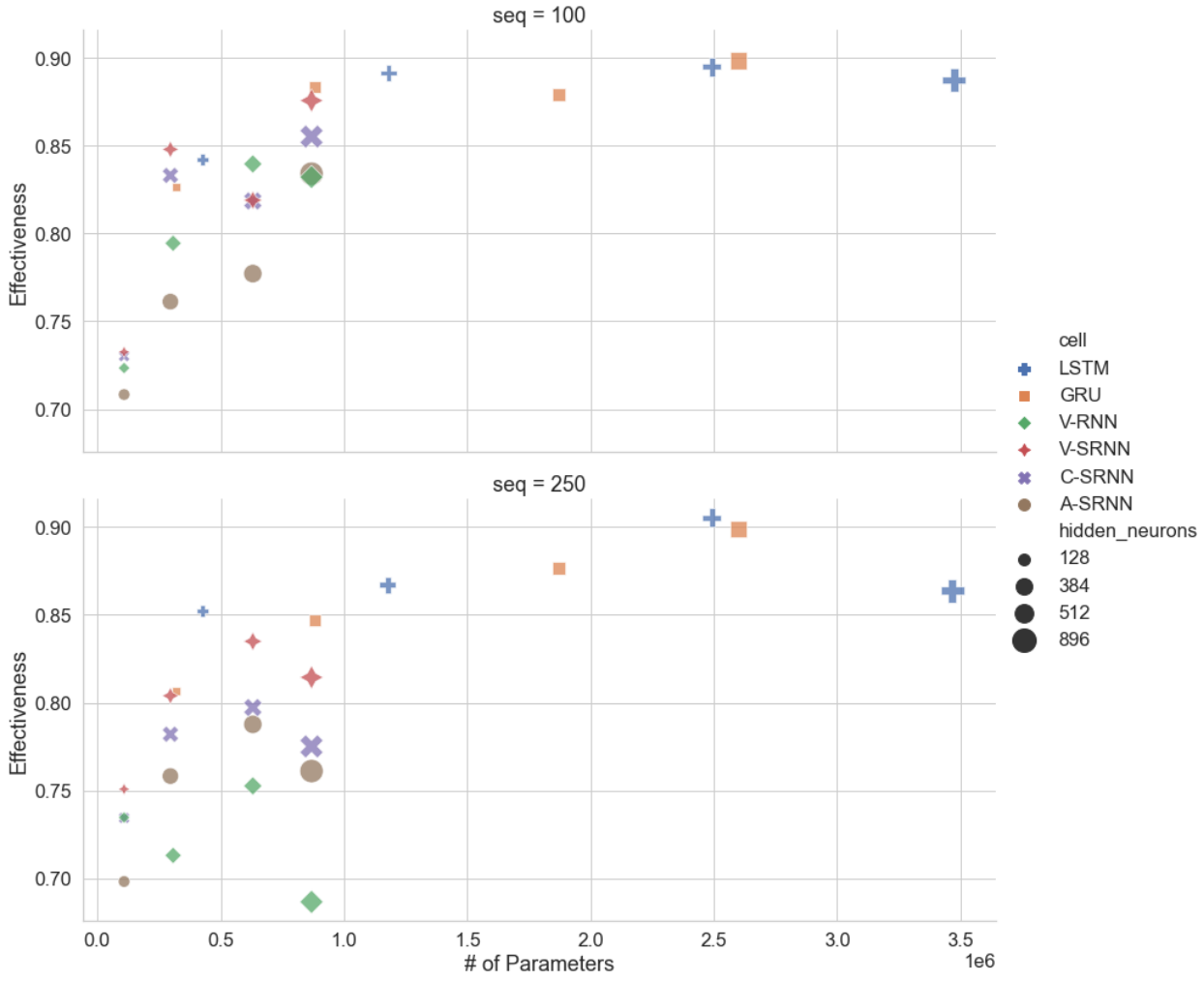
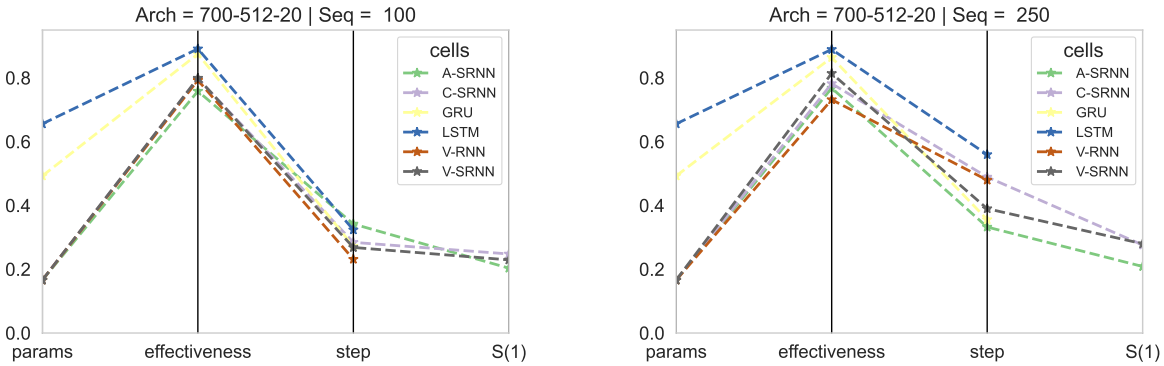


Figure 5.7: Effectiveness versus number of parameters of SRNNs with heuristic method and traditional RNNs in the classification of the SHD test data set.

The radar graph shown in Figure 5.8 illustrates the convergence steps, effectiveness, num-

ber of parameters (invariant to the sequence length) and spikes per layer (only for spiking models) of SRNNs and traditional RNNs with the *700-512-20* architecture. The above mentioned metrics are averaged and they represent the point in time at which the maximum effectiveness was reached for the models. In general, for this particular architecture, the LSTM model obtained the best performance, however, it has more parameters than the other models.

For a 100-step sequence, all spiking models presented a similar performance in terms of effectiveness and number of spikes (S(1)). All spiking models have three times less parameters than GRU and four times less than LSTM. The step metric suggests that almost all spiking models converged in a similar way. For a 250-step sequence, the spiking models performed differently in their metrics. In this case, the S(1) metric suggests that the C-SRNN and V-SRNN models have more spikes than the A-SRNN model. The step metric suggests that the convergence of the A-SRNN model was similar to that of the GRU and V-SRNN model. Finally, the effectiveness reported in this dataset version were similar to those of the 100-step sequence version.



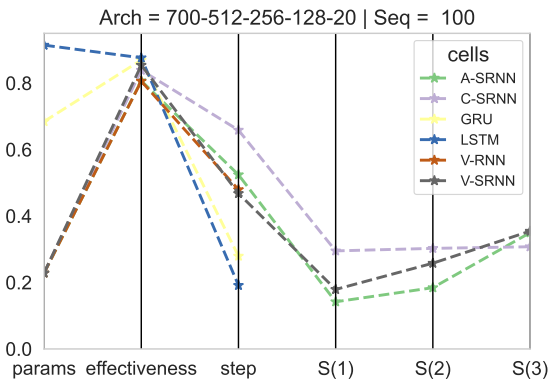
a) 10ms step simulation (100 size steps)

b) 4ms step simulation (250 size steps)

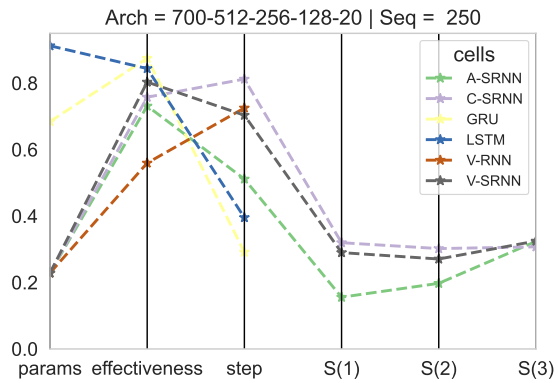
Figure 5.8: Parallelogram chart with the effectiveness, convergence steps, parameters, and spikes per layer (only for the latter spiking models) of SRNNs and RNNs using the 700-512-20 architecture.

The latter radar graph shown in Figure 5.9 presents the performance of spiking and non spiking models using the *700-512-256-128-20* architecture. Some models, such as C-SRNN, converge slower than others. Regarding the number of learnable parameters, GRU and LSTM required more parameters than the other models, however, the effectiveness of these models is not much higher than spiking models. The previous analysis is independent of the sequence length version.

Concerning the 100-step length version, the LSTM model obtained the best score in effectiveness and convergence steps, however, this model required more parameters than the other models. With regard to spiking models, V-SRNN obtained on average better effectiveness and convergence than the other spiking models. For the 250-step length version, GRU model achieved the best score in effectiveness and convergence steps, and also required fewer parameters than LSTM model. Regarding the spiking effectiveness and number of spikes per layer, V-SRNN obtained better effectiveness than A-SRNN and C-SRNN, however, A-SRNN had fewer spikes in its layers.



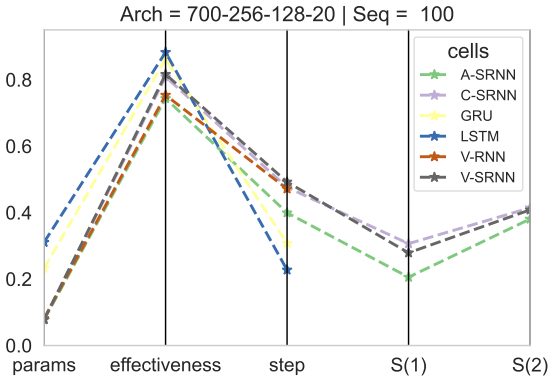
a) 10ms step simulation (100 size steps)



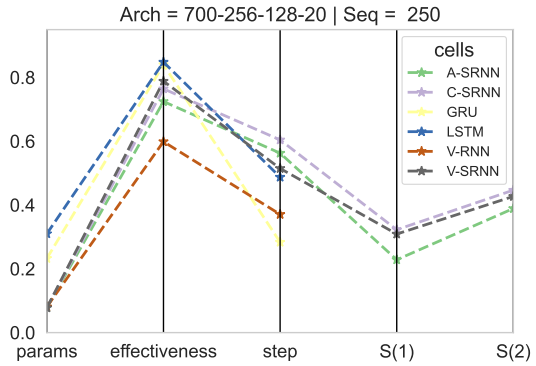
b) 4ms step simulation (250 size steps)

Figure 5.9: Parallelogram chart with the effectiveness, convergence steps, number of parameters, and spikes per layer (only for the latter spiking models) of SRNNs and RNNs with the 700-512-256-128-20 architecture.

The radar graph shown in Figure 5.10 shows the performance of the models using the *700-256-128-20* architecture. The results are similar to those shown in Figure 5.9. For both datasets, C-SRNN convergence was slower than the others models; LSTM models obtained the best effectiveness; and A-SRNN had fewer spikes than the other spiking models.



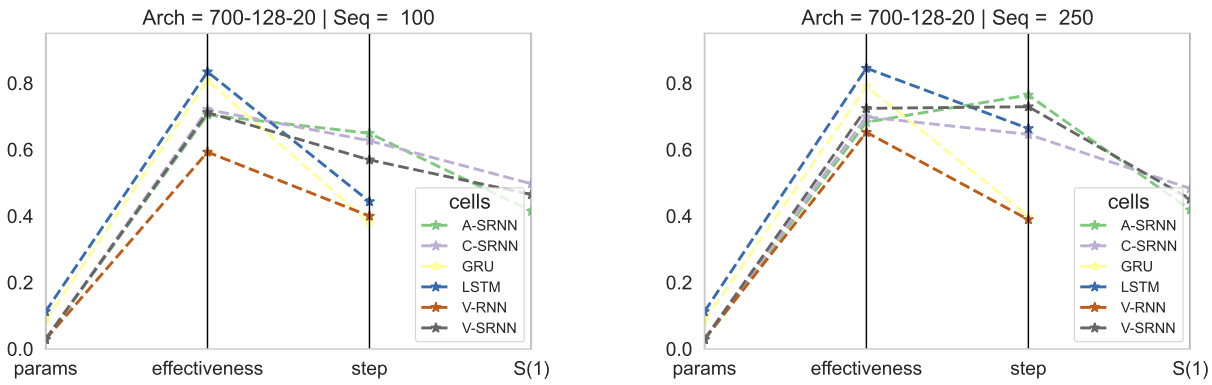
a) 10ms step simulation (100 size steps)



b) 4ms step simulation (250 size steps)

Figure 5.10: Parallelogram chart with the effectiveness, convergence steps, parameters, and spikes per layer (only for spiking models) of SRNNs and RNNs with the 700-256-128-20 architecture.

The radar graph shown in Figure 5.11 reports the results of the *700-128-20* architecture, we can see that V-RNN obtained the worst effectiveness, nevertheless, it had one of the best time convergences on average. In relation to the number of spikes, V-SRNN and C-SRNN models had the highest number of spikes per layer when reaching their max effectiveness value.



a) 10ms step simulation (100 size steps)

b) 4ms step simulation (250 size steps)

Figure 5.11: Parallelogram chart with the effectiveness, convergence steps, number of parameters, and spikes per layer (only for the latter spiking models) of SRNNs and RNNs with the 700-128-20 architecture.

5.2.1 Discussion

The aim of this discussion is to compare the results obtained with the models studied for solving the SHD problem. First, we discuss about the effectiveness reached by the models. Second, we expand an analysis about the number of learnable parameters (weights). Finally, we present the relation between effectiveness and number of spikes per layer for each spiking model.

Effectiveness

In the current study, a fair comparison[1] of SRNNs and traditional RNNs showed that the effectiveness have similar performances, especially between V-SRNN and GRU models. A possible explanation may be the external memory underlying these models. Another possible explanation is an appropriate use of their gradients and setting of hyperparameters.

Learnable Parameters

The results of this study indicate that spiking models with an appropriate number of parameters perform similarly to gated RNNs for the SHD dataset. Surprisingly, there is no significant difference in effectiveness when learnable parameters reach a certain value. The analyses on the number of learnable parameters are also consistent with the sequence length, however, the results can not be extrapolated to all sequences because gradient problems could arise.

In response to our first research question **RQ1**: *What is the minimum number of parameters needed to solve the classification tasks used in this work, regardless of the RNN or SRNN architecture?*, our findings suggest that the models require at least 1M parameters for solving the SHD problem, regardless of their architectures.

[1]Considering the number of learnable parameters, among other training strategies.

Spikes

Our results show that the number of spikes is related to the convergence steps of the model when the heuristic method is used. One unanticipated finding is that the last layer in different SRNNs has the same number of spikes on average. A possible explanation for this might be that the last layer generally is used for classification and the others layers (S(1), ... , S(L-1)) for data representation.

With regard to single-layer architecture spiking models, the architecture with 128 neurons in the hidden layer needs more spikes than the one with 512 neurons, which could be related to the threshold value. However, the hidden layers with 128 and 512 neurons in multilayer architectures presented fewer spikes than in the single-layer architecture. Thus, this could also relate to the capability of the model based on the number of learnable parameters. Concerning the three-layer SRNN architecture, the number of spikes in S(1) was similar to that of the single-layer architecture with 512 neurons. These results are possibly due to the ϑ and u_{rest} values used. This phenomenon can be seen also for S(2) of the three-layer SRNN and S(1) of two-layer SRNN architectures.

5.3 Literature fixed Parameters vs Heuristic Method

The objective of this section is to compare the performance between literature fixed and heuristic parameters definition in the SRNN models. This comparison allowed us to find the best spiking model and its architecture solving the SHD problem. We applied the permutation test (Eq. 3.17) to each experiment. The heat map in Figure 5.12 shows the performance of spiking models with literature fixed parameter (left) and heuristic parameter (right) definitions, for two different length sequences (upper and bottom). In general, the heuristic method achieved the best effectiveness in almost all architectures, regardless of the time step simulation. Among these spiking models, V-SRNN obtained the best performance.

Figure 5.12: Effectiveness of SRNN models with literature fixed and heuristic parameters for the SHD problem.

For the 100-step length, the heuristic method increased the performance of the models compared with respect to the literature fixed baseline. This increase is more notable in the three-layer architecture of V-SRNN followed by the C-SRNN and A-SRNN models. This order is kept in the single-layer model with 512 neurons and in the 2-layer model. In contrast, the single-layer model with 128 neurons obtained a similar performance in all models. For the

250-step length, the V-SRNN with heuristic parameters reported the best effectiveness. In this scenario, almost all architectures dropped down their performance, especially C-SRNN and A-SRNN with three hidden layers. However, the single-layer architectures kept their performance.

The heat map shown in Figure 5.3 outlines the p-values (colour-bar) in three SRNN models (axis x) and four architectures (axis y). These p-values arise from the statistical comparison of means between SRNNs trained with literature fixed parameters and SRNNs trained with parameters obtained from the heuristic method. Each p-value represents one experiment with 5 runs each. In general, most p-values are less than 0.05 which suggests that there is an improvement by applying the heuristic method to compute SRNNs.

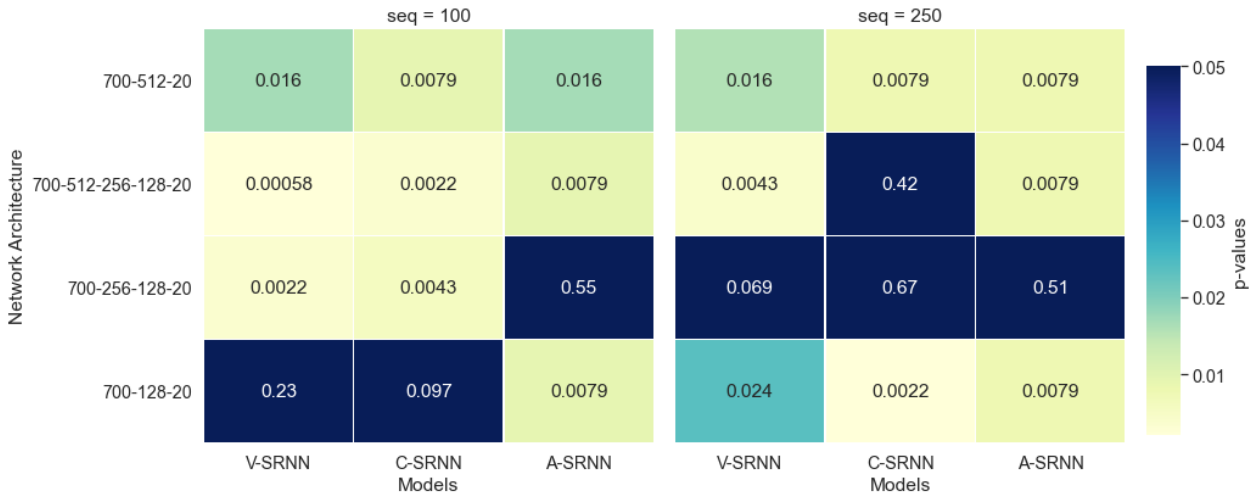


Figure 5.13: P-values of the permutation test between literature fixed and heuristic parameters of SRNNs.

5.3.1 Discussion

The present study found that the heuristic computation method makes the training of SRNNs more stable and also improves the performance. Another interesting finding was that SRNN with literature fixed parameters and architecture configuration taking from [4], [14], [70] does not improve with respect to the heuristic method. A possible explanation for such results may be that the above-mentioned authors set the optimal hyperparameter values to their architecture. It is probable therefore that arbitrary literature fixed parameter worked only for their experiments.

Previous studies related to adaptive threshold [4], [70], assume that this approach is better than the static threshold. However, we report greater effectiveness using a well-calculated static threshold in multilayer V-SRNNs. A possible explanation for this is that A-SRNN is harder to train and might need another way to compute the initial threshold value.

In reference to the hypothesis **H1**, which claims that *the SRNN models with the hyperparameters calculated using a heuristic method can obtain a higher accuracy than the SRNN*

models using literature fixed model parameters, we obtained statistically significant improvement by applying the heuristic method to compute SRNNs, so **H1** is accepted solving the SHD Problem.

Chapter 6

Memory and Optimisation Criterion

In this chapter, we report a fair evaluation between the proposed Gated SRNN (G-SRNN) and GRU model. The G-SRNN is a variant of the V-SRNN, which proved to have better effectiveness than the other spiking models for solving the SHD problem. In order to compare other aspects of these models, we also analyse the convergence step and number of parameters. Finally, we compare the effectiveness of these models using a statistical test.

In reference to the optimisation criterion, we evaluate the space information bottleneck on V-SRNNs with three-layer architecture (700-512-256-128-20) using only 100-step sequence. We use this particular architecture of V-SRNN because it reported the best accuracy (effectiveness) for solving the SHD problem. In addition, we analyse the numbers of spikes per layer between different versions of mutual information estimators and the Lagrange values γ.

6.1 Two-Level External Memory

We conducted some experiments and comparisons between the G-SRNN and GRU models. This comparison allowed us to have spiking models with the same number of parameters for some specific architectures. These models were trained using the proposed experimental framework, excluding the multilayer networks. We evaluated single-layer architectures because a single-layer GRU model achieved a good performance compared to the multilayer architectures with a larger number of parameters. Finally, we analysed the performance of these models using a statistical test.

The heat map shown in Figure 6.1 outlines the effectiveness (color-bar), network architecture (y-axis) and spiking and non-spiking gated models (x-axis). In general, the G-SRNN and GRU reported similar effectiveness in both architectures and sequence length. In addition, Figure 6.1 shows that G-SRNN effectiveness varies more than GRU, especially in the smaller networks.

The results for both sequence lengths of the dataset were similar. G-SRNN effectiveness was on average slightly lower than GRU model with 512 hidden neurons, but for the architecture with 128 neurons in the hidden layer the difference in performance was larger in favour of GRU. There is a clear trend to decrease effectiveness when they have fewer neurons in the

hidden layer.

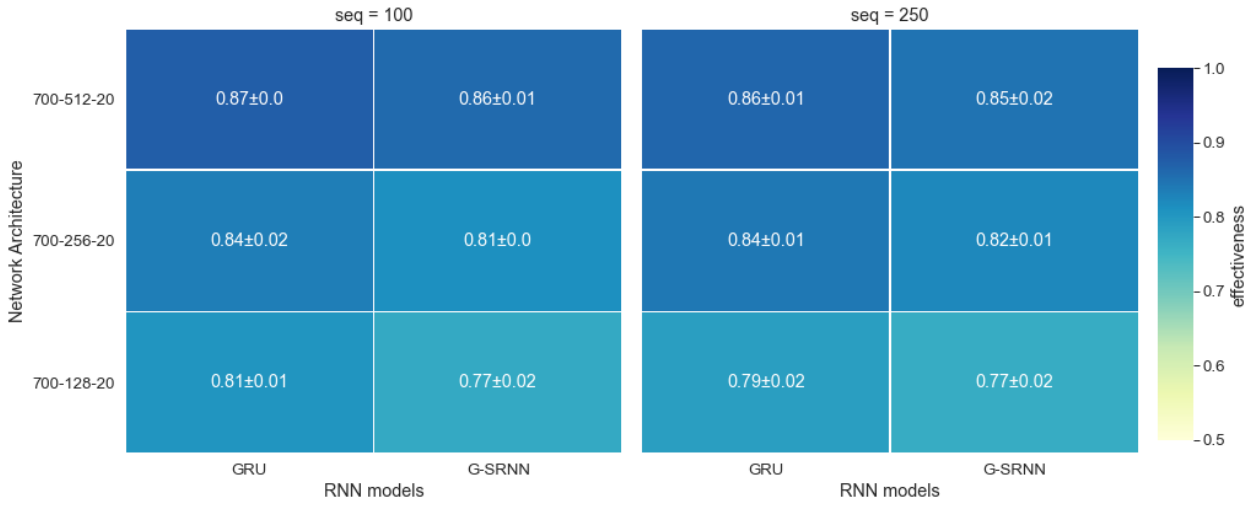


Figure 6.1: Gated-SRNN and GRU models performance for the SHD classification problem.

The heat map shown in Figure 6.2 presents the p-value (color-bar), network architecture (y-axis) and spiking vs non-spiking gated networks (x-axis). This figure compared the effectiveness between G-SRNN and GRU using the permutation test (Equation 3.17). In general, almost all architectures obtained a p-value higher or close to 0.05 which means that there were no significant differences between both models.

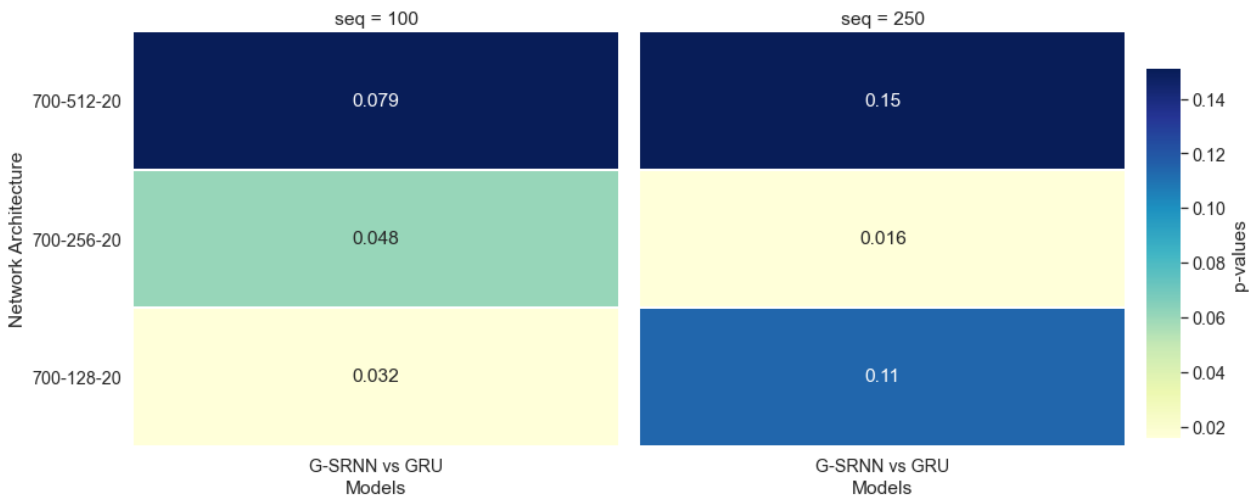
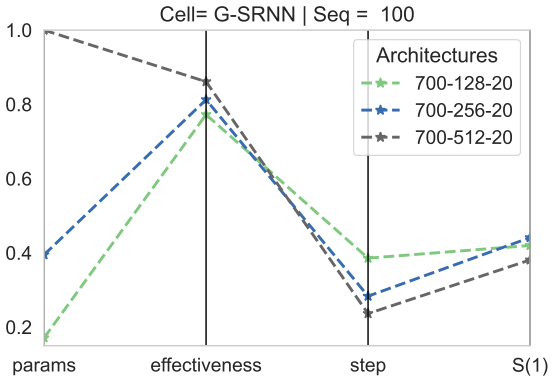


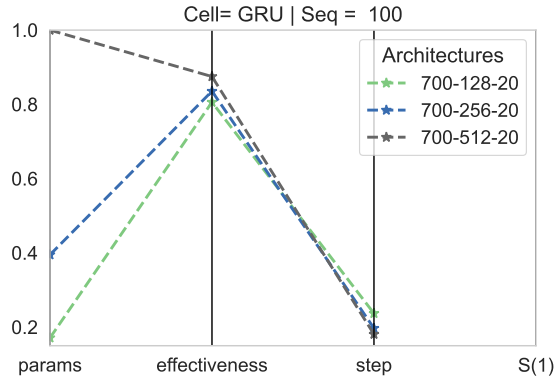
Figure 6.2: Gated-SRNN vs GRU effectiveness comparison using permutation statistical test for significance

The radar chart shown in Figure 6.3 outlines the effectiveness, convergence steps, parameters, and spikes per layers (only for the latter spiking models) in different architectures of G-SRNN (left) and GRU (right) and sequence lengths. Overall, G-SRNN and GRU reported similar effectiveness independent of the sequence length data. In addition, the GRU architectures reported similar number of convergence steps for both sequence lengths.

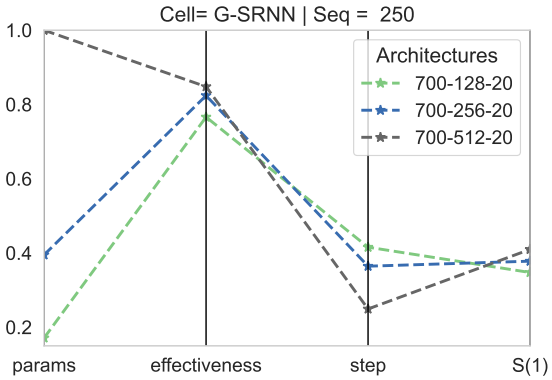
Concerning architectures, GRU and G-SRNN with 512 neurons in its hidden layer converged faster, in terms of number of iterations, than the architectures with 256 and 128 neurons, but demanded more parameters. In addition, G-SRNN was slower to converge than the GRU network. Interestingly, the trained architectures behaved similarly in both versions of sequence length in terms of effectiveness. In reference to the number of spikes, G-SRNN for 100-step sequence needed more spikes than for 250-step sequence.



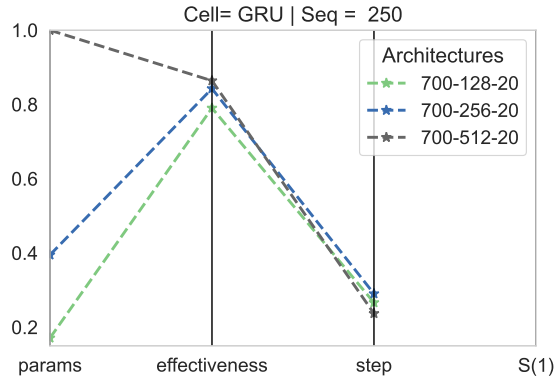
a) G-SRNN performance metrics using 10ms step simulation (100 size steps)



b) GRU performance metrics using 10ms step simulation (100 size steps)



a) G-SRNN performance metrics using 4ms step simulation (250 size steps)



b) GRU performance metrics using 4ms step simulation (250 size steps)

Figure 6.3: Radar charts with the effectiveness, convergence steps, parameters, and spikes per layers (only for the latter spiking models) of G-SRNN and GRU architectures.

6.1.1 Discussion

As mentioned in the literature review, unfair comparisons between spiking and non-spiking models are generally conducted. This part of the study aimed to assess those models with a similar number of hidden neurons and learnable parameters (weights). We found that both models perform similarly when they are evaluated on similar conditions. This could be because both models (GRU and G-SRNN) belong to the category of the *Infinite Impulse Response* (IIR) systems [57]. Furthermore, it can be seen that the spiking model is a natural representation of an IIR (because it uses discrete pulses in its hidden layers), and makes sense that both models have similar performance.

The present findings seem to be consistent with other works which have found that gated RNNs can fix the gradient problems. Our results suggest that the results with 250- and 100-step sequences are statistically similar with G-SRNN and GRU. In addition, the gradient problem is more severe in SRNN models without gating mechanism or external memory approach.

In reference to the hypothesis **H2**, which states that *the SRNN models using binary values in their layers have similar accuracy with respect to the traditional RNNs (such as GRU) using continuous values, solving the SHD problem*, we obtained that in almost all architectures and sequence versions, there is no statistical differences between GRU and G-SRNN effectiveness classifying the SHD problem. Therefore **H2** is accepted, solving the SHD problem.

The reported results must be interpreted with caution because only one dataset was tested. In future investigations, it might be possible to use a different and more complex dataset in order to generalise our findings. Nevertheless, we conjecture that these results would be kept for other binary spiking datasets.

6.2 Space Information Bottleneck on V-SRNN

Based on the proposed methodology, we evaluated the space IB as an optimisation criterion in the best architecture of V-SRNN (*700-512-256-128-20*). We first present the effectiveness of the model using different estimators of the mutual information and different values of γ (Eq. 3.35). In addition, we present the iteration steps that were performed to reach the maximum effectiveness. Then, we analyse the results of the average number of spikes per layer. Finally, we use a statistical test to answer the research question.

Figure 6.4 compares the effectiveness (top heat map) and iteration steps (bottom heat map) of the V-SRNN model with respect to mutual information estimator (x-axis) and γ values (y-axis). Overall, mutual information estimators do not affect the effectiveness of the model, however, there are some changes in the number of convergence steps.

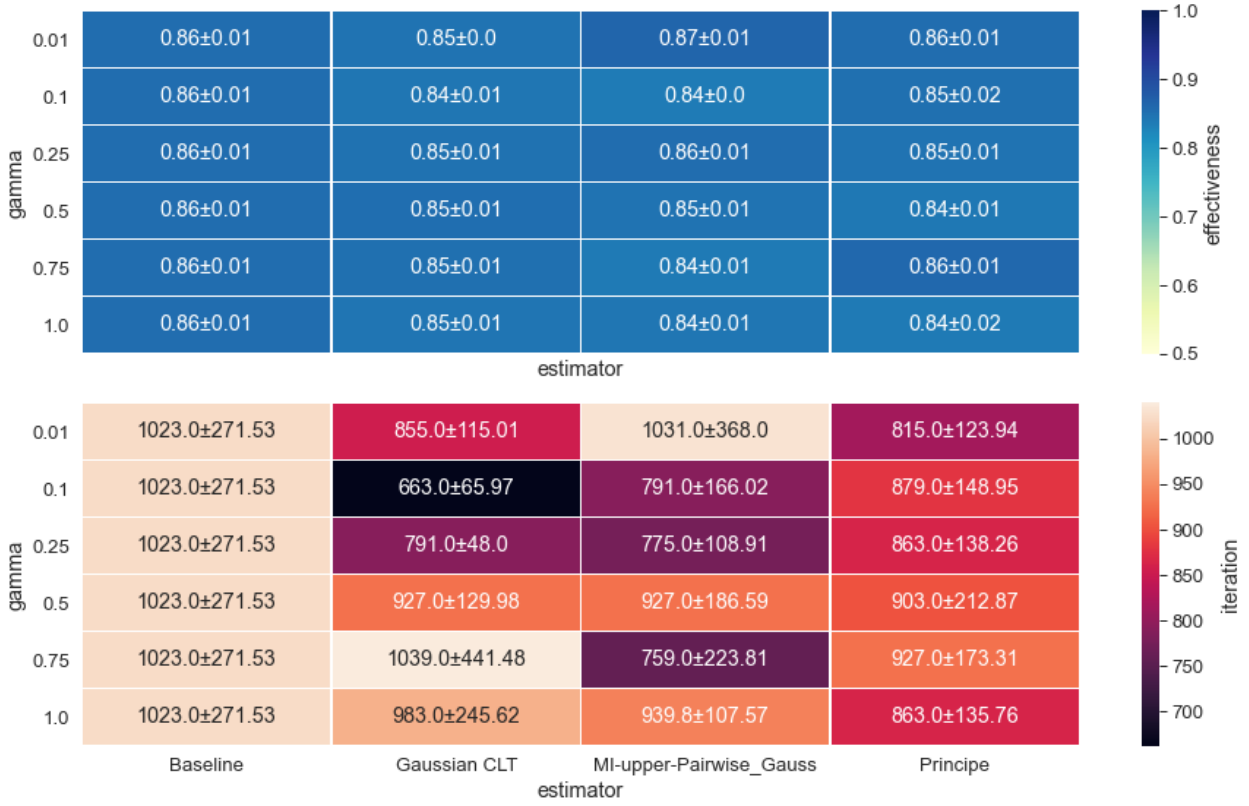


Figure 6.4: Effectiveness and iteration steps of V-SRNN with the 700-512-256-128-20 architecture using IB in the classification of the SHD dataset.

With reference to the Gaussian estimator based on the Central Limit Theorem (CLT), the effectiveness for almost all values of γ maintained a similar effectiveness. Interestingly, there were differences in the number of iteration steps, since as the values of *gamma* increased, the iteration steps increased on average. Finally, a comparison of the two metrics (effectiveness and iteration steps) revealed that using pairwise Gauss estimator with a $\gamma = 0.01$ achieved slightly better results on average.

Concerning upper pairwise Gauss estimators, one of the results obtained the highest effectiveness between the baseline and the others estimators (87% of accuracy). In contrast,

the same estimator obtained the lowest effectiveness values on average between all estimators (84% of accuracy). In general, almost all γ values had similar number of iteration steps.

With respect to Principe's estimator, the effectiveness and iteration steps were similar among almost all the values of γ. Moreover, the performance of this estimator was more affected when the values of γ were 0.5 and 1.0 (84% of accuracy).

The heat map shown in Figure 6.5 outlines the p-values of comparing Cross Entropy (Baseline) and Information Bottleneck (with different estimators of mutual information and γ values) in terms of effectiveness. Overall, no significant differences in almost all experiments were found. Only in one particular case, IB negatively affected the performance of the model because the p-value is under 0.05. On the other hand, the heat map shown in Figure 6.6 presents the p-values in terms of iterations steps. Similarly to the previous analysis, no significant differences were found between Cross Entropy and Information Bottleneck.

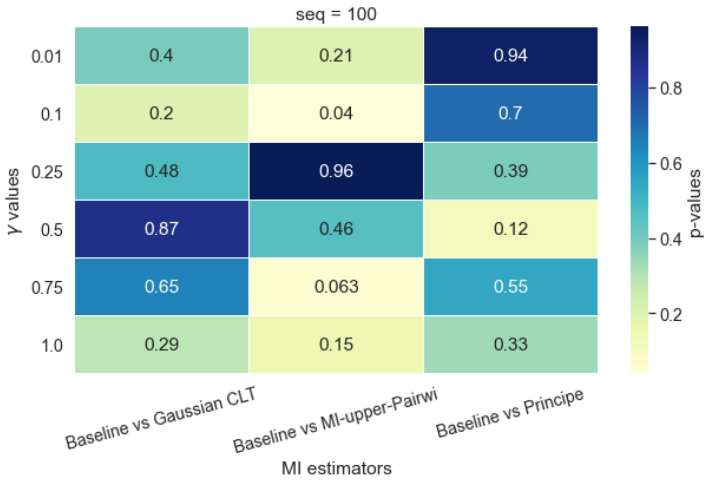


Figure 6.5: Statistical permutation test between Cross Entropy and Information Bottleneck in terms of effectiveness.

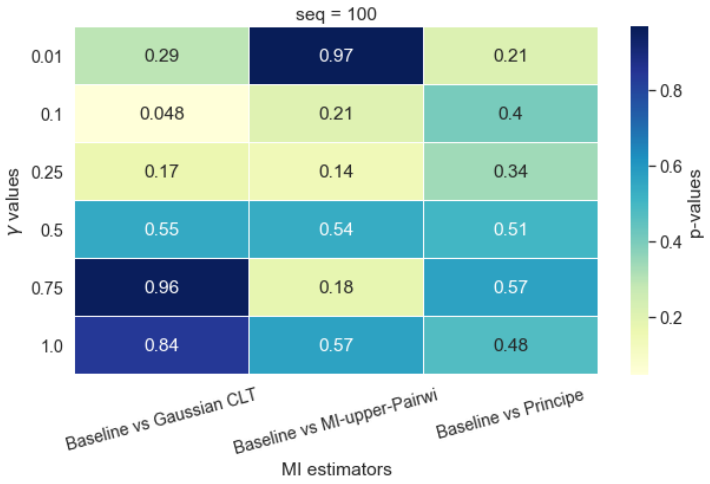


Figure 6.6: Statistical permutation test between Cross Entropy and Information Bottleneck in terms of iterations steps.

The parallelogram chart shown in Figure 6.7 outlines the proportion of the number of spikes (y-axis), γ values (y-axis), and estimators (color) when the model reached the best effectiveness in the validation set. In general, the baseline optimisation criterion (Cross-Entropy) obtained more spikes than Information Bottleneck with different estimators. Additionally, when $\gamma = 1$ almost all estimators had a similar number of spikes per layer, except for Principe's estimator.

In almost all cases the baseline has on average more spikes than IB estimators in layer one $S(1)$. Focusing on layer two $S(2)$, an elbow emerged for all γ values and estimators, which is more pronounced in IB with Principe's estimator. Finally, in layer three $S(3)$, all optimisation criteria, including the baseline, did not have a defined behaviour because they resulted in different numbers of spikes.

The single most striking observation to emerge from the spikes comparison was that Gauss CLT and MI-upper-Pairwise-Gauss estimators reported similar number of spikes per layers. This is interesting because our proposed Gaussian CLT estimator is more simple (it considers the number of samples and neurons as a single component) than MI-upper-Pairwise-Gauss (which considers the number of samples and neurons as two different components).

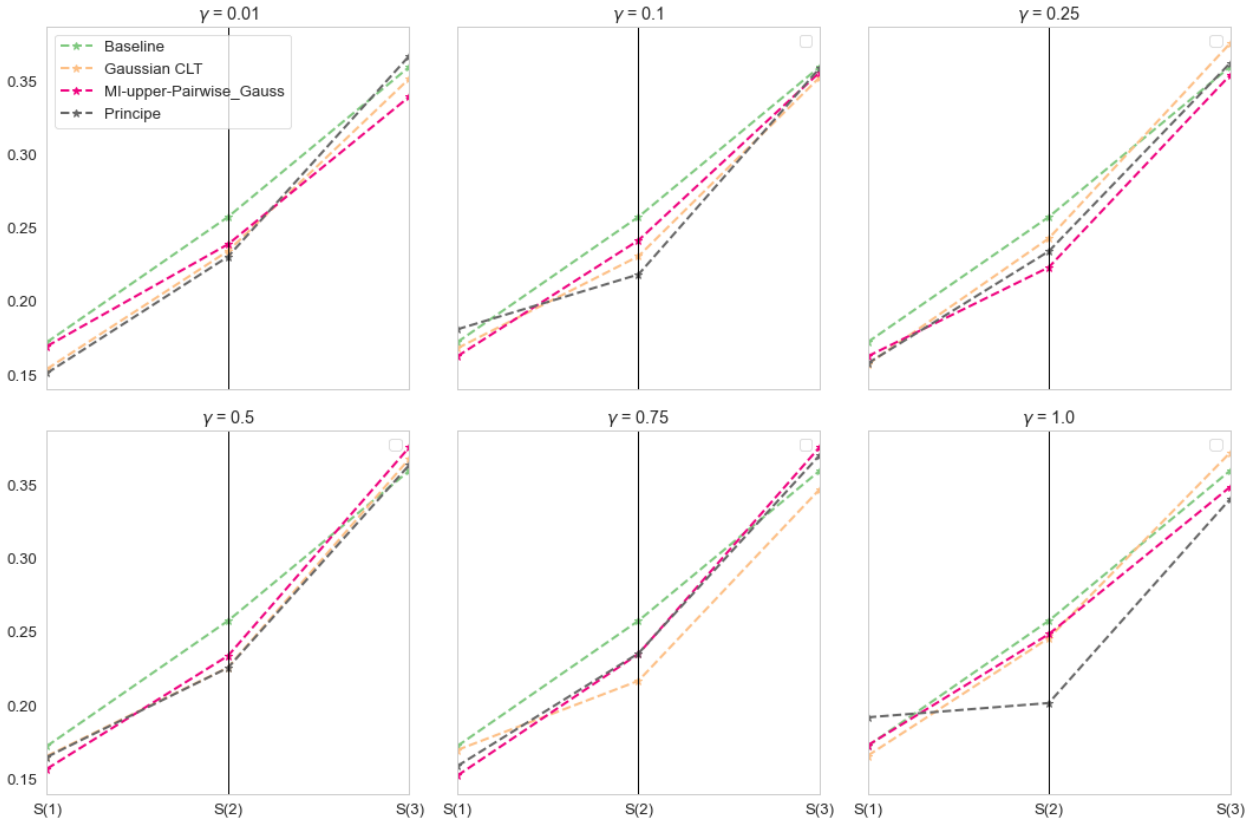


Figure 6.7: Parallelogram of the average number of spikes per layer when the model achieved the best effectiveness value in the validation set.

6.2.1 Discussion

The current study found that space information bottleneck does not improve the performance of multilayer V-SRNN. It is difficult to explain this result, but it might be related to the formulation of IB that only considered the space but not the time. Thus, in order to improve the performance in this model, space and time may be required. Another possible explanation for this is that estimators of mutual information could fail when using discrete random variables.

In some cases, the estimators need a normalised random variable according to their definition. In our evaluation, we conducted our experiments normalising the batch of binary values for Principe and MI-upper-Pairwise-Gauss estimators. This normalisation only makes sense if the mean has a meaning. According to the Maximum Likelihood Estimation, the mean represents the probability of positive events (or spikes) of a Bernoulli random variable. Therefore, normalising our random variable is justified.

In response to our last research question **RQ3**: *Does the optimisation criterion space Information Bottleneck improve the performance of SRNNs compared to those trained with Cross-Entropy?*, our findings suggest that there was no statistically significant difference between models training with cross-entropy and space information bottleneck solving SHD problem.

Based on the results obtained in terms of effectiveness, two important findings emerge from our experiments. The first one is related to the effectiveness since the mutual information in the IB method works as a regularisation that could prevent the overfitting of the model. The second one is related to the average number of spikes generated per layer, where models trained with IB on average have fewer spikes than those trained with CE.

In future investigations, it might be possible to use space and time Information Bottleneck. However, it is important to note that the time component will increase the computational cost of the mutual information computation.

Chapter 7

Conclusions

Our findings suggest that SRNN and traditional RNN can be trained and evaluated using the proposed experimental framework. Our experimental framework allows a complete analysis of the relationship of the studied models. In fact, our approach can be applied to any deep learning model. As a result, we generated a baseline for different models and architectures in the classification of two binary datasets varying their sequence length.

One of the most significant findings to emerge from this study is that a heuristic method can be used to estimate the parameters of multilayer SRNNs to improve their performance solving the SHD problem. In addition, this study demonstrated, for the first time, that SRNN model parameters are related to each other and that a good parameter selection can alleviate the gradient problems.

To the best of our knowledge, most binary models have a lower performance than continuous models. However, our results suggest that binary RNNs inspired by biological concepts perform similarly to traditional RNNs solving SHD problem, which is an important finding. The main conclusions obtained in the various aspects evaluated in this work are presented in what follows.

Parity Bit and SHD Baselines

Parity bit problem can be empirically simulated with a proper number of parameters, regardless of the RNN model, architecture and sequence length. In addition, LSTM, GRU, C-SRNN and V-SRNN model validation holds an invariant accuracy in almost all architectures and sequence lengths. Our results in the classification of the parity bit dataset empirically support that SRNNs are Turing complete.

This study has found that, in general, traditional RNNs performs better than SRNN models with parameters defined for a specific architecture for classifying the SHD dataset. Despite the empirical nature of this work, it offers some insight about design of SRNNs and other aspects that need to be taken into account for gradients. The generalisability of these results is subject to certain limitations, such as the fact that we only used one dataset.

SRNN Model Parameters and Weights (Learnable Parameters)

This study was aimed to determine that SRNN models parameters are closely related to each other. In addition, one of the most significant findings to emerge is that a correct selection of the parameters could produce effective models because these parameters can also control the behavior of the gradients and consequently accelerate the learning convergence time. Although this study did not confirm the relationship between weights and effectiveness, it did partially corroborate that as more parameters are added, models improve their performance.

Heuristic Method for SRNN Parameters Computation

In this work we designed methods to evaluate and improve SRNNs models. These findings suggest that in general a good parameter estimation improves the SRNNs performance, especially in multilayer architectures. One of the most significant findings is that the three-layer V-SRNN model has similar performance than GRU model in the classification of the SHD dataset. Our evidence suggests that heuristic parameters stabilise the training procedure and improve the model performance.

In addition, the results of this research support the idea that small threshold ϑ and u_{rest} saturate the hidden layer with too many spikes, and although they produce a faster convergence, the solution is generally not good. In contrast, a large ϑ and u_{rest} produce fewer spikes in some cases, which slows down the convergence of the model and can lead to a sub-optimal solution.

Finally, a number of important limitations need to be considered. First, heuristic method for single-layer architectures needs to consider better assumptions (for instance considering that one third of the spikes events are available for sampling). Second, the range of optimal values of $\tilde{n}$ changes as the time sequence becomes large. Finally, it is recommendable to use the surrogate gradient function used in this work for a stable model training using our heuristic method.

Two-Level External Memory in V-SRNN

This study has shown that SRNNs could be used as gated RNN with similar effectiveness as the GRU under certain conditions. It also contributed to improving our understanding of SRNNs with external memory and their gradient flow. In summary, our evidence suggests that SRNNs are a natural expression of Infinite Impulse Response because Leaky Integrate and Fire model is based on Dirac Delta function. Despite the exploratory nature of this study, it offers some insight into the spiking and non spiking models classifying binary datasets evaluated in similar conditions.

Space Information Bottleneck in V-SRNN

Returning to the question posed at the beginning of this study about IB, it is now possible to state that in our experiments there are no significant improvements in terms of effectiveness using it spatially. However, there is a kind of compression in the layers that show the optimisation criterion is fulfilling its function of generating compressed representations.

One of the most significant results emerging in this aspect is that a simple estimator based on the central limit theorem and differential mutual information is statistically similar to the Principe and MI-upper-Pairwise-Gauss estimators in terms of effectiveness. When analysing the convergence steps, the former estimator reported on average the best convergence values using the spatial information bottleneck.

7.1 Final Comments and Future Work

This work focused on improving the performance of SRNNs by using training strategies and methods widely used in the deep learning community. The application of these strategies and methods helped to create new models and ways to compute the models parameters in order to improve their performance compared to our baselines. In particular, we found that these models of discrete nature can achieve competitive results with respect to GRU with fewer parameters solving the SHD problem.

For future work, we want to test our heuristic method and the model with external memory on more datasets. In addition, we would like to improve the adaptation of the bottleneck principle in SRNNs considering space and time. Finally, we will formally prove that our heuristic method is equal or better than defining a model parameter arbitrarily, and then we will implement these models on specialised SNN hardware.

Bibliography

[1] Ahmed Abusnaina and Rosni Abdullah. Spiking neuron models: A review. *International Journal of Digital Content Technology and its Applications*, 8:14–21, 06 2014.

[2] Charu C Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer, 2018. ISBN 978-3-319-94463-0.

[3] Coryn Bailer-jones, David Mackay, and Philip Withers. A recurrent neural network for modelling dynamical systems. *Network: Computation in Neural Systems*, 9, 08 2002. doi: 10.1088/0954-898X_9_4_008.

[4] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 795–805, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[6] Lars Buesing and Wolfgang Maass. A spiking neuron as information bottleneck. *Neural Comput.*, 22(8):1961–1992, August 2010. ISSN 0899-7667. doi: 10.1162/neco.2010.08-09-1084.

[7] Anthony Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95:1–19, 08 2006. doi: 10.1007/s00422-006-0068-6.

[8] Ivan Chelombiev, Conor Houghton, and Cian O'Donnell. Adaptive estimators show information compression in deep neural networks. In *International Conference on Learning Representations*, 2019.

[9] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[10] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Neural Information Processing Systems, Workshop on Deep Learning*, 2014.

[11] Iulia M. Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *IEEE International Conference on Acoustics, Speech and Signal*

Processing, pages 8529–8533, 2020.

[12] Rui Ponte Costa, Yannis M. Assael, Brendan Shillingford, Nando de Freitas, and Tim P. Vogels. Cortical microcircuits as gated-recurrent neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 271–282. Curran Associates Inc., 2017. ISBN 9781510860964.

[13] Thomas Cover and Joy Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006. ISBN 0471241954.

[14] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2020.

[15] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27(4):1071–1092, Sep 2020. ISSN 1886-1784. URL `https://doi.org/10.1007/s11831-019-09344-w`.

[16] Jeffrey Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[17] Bert Fristedt and L. Gray. A modern approach to probability theory. In *A modern approach to probability theory*, 1996.

[18] Abbas El Gamal and Young-Han Kim. *Network Information Theory*. Cambridge University Press, USA, 2012. ISBN 1107008735.

[19] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999.

[20] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. ISBN 1107635195.

[21] G.J. Gibson, S. Siu, and C.F.N. Cowen. Multilayer perceptron structures applied to adaptive equalisers for data communications. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 1183–1186 vol.2, 1989.

[22] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 1756–1760, 08 2013.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

[24] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL

http://arxiv.org/abs/1410.5401. cite arxiv:1410.5401.

[25] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[26] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition, 1998. ISBN 0132733501.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1026–1034, USA, 2015. IEEE Computer Society. ISBN 9781467383912. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[29] A. L. HODGKIN and A. F. HUXLEY. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117 (4):500–544, Aug 1952. ISSN 0022-3751. URL https://pubmed.ncbi.nlm.nih.gov/12991237. 12991237[pmid].

[30] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with LIF neurons. *CoRR*, abs/1510.08829, 2015. URL http://arxiv.org/abs/1510.08829.

[31] Sander M Bohte Hélene Paugam-Moisy. *Computing with Spiking Neuron Networks*, pages 335–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-540-92910-9.

[32] Melissa Johnson and Sylvain Chartier. Spike neural models part ii: Abstract neural models. *The Quantitative Methods for Psychology*, 14, 02 2018. doi: 10.20982/tqmp.14.1.p001.

[33] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[34] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, San Juan, Puerto Rico, 2016. ICLR.

[35] Q. Kang and W. Guo. 4 - biomimetic smart nanopores and nanochannels. In Mario Tagliazucchi and Igal Szleifer, editors, *Chemically Modified Nanopores and Nanochannels*, pages 85–102. William Andrew Publishing, Boston, 2017. ISBN 978-0-323-40182-1. doi: https://doi.org/10.1016/B978-0-323-40182-1.00004-X. URL https://www.sciencedirect.com/science/article/pii/B978032340182100004X.

[36] Artemy Kolchinsky and Brendan D. Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7), 2017. ISSN 1099-4300. doi: 10.3390/e19070361. URL https:

//www.mdpi.com/1099-4300/19/7/361.

[37] Artemy Kolchinsky, Brendan D. Tracey, and David H. Wolpert. Nonlinear information bottleneck. *Entropy*, 21(12), 2019. ISSN 1099-4300. URL https://www.mdpi.com/1099-4300/21/12/1181.

[38] Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. In *Conference on Neural Information Processing Systems*. NeurIPS, 2020.

[39] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016. ISSN 1662-453X.

[40] Chunhung Li Li and Leem C. Minimum cross entropy thresholding. *Pattern Recognition*, 26(4):617–625, 1993. ISSN 0031-3203. doi: https://doi.org/10.1016/0031-3203(93)90115-D. URL https://www.sciencedirect.com/science/article/pii/003132039390115D.

[41] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4313–4324. PMLR, 26–28 Aug 2020. URL http://proceedings.mlr.press/v108/li20j.html.

[42] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJg8TeSFDH.

[43] Qiong Liu and Ying Wu. *Supervised Learning*, pages 3243–3245. Springer US, Boston, MA, 2012. ISBN 978-1-4419-1428-6. URL https://doi.org/10.1007/978-1-4419-1428-6_451.

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[45] Ying Ma and Jose C. Principe. A taxonomy for neural memory networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(6):1780–1793, 2020.

[46] Walter Marcotti and Sergio Masetto. *Hair Cells*, chapter 1. American Cancer Society, 2010. ISBN 9780470015902. doi: https://doi.org/10.1002/9780470015902.a0000181.pub2. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470015902.a0000181.pub2.

[47] Laura Martignon. Information theory. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 7476 – 7480. Pergamon, Oxford, 2001. ISBN 978-0-08-043076-8. doi: https://doi.org/10.1016/

B0-08-043076-7/00608-2. URL http://www.sciencedirect.com/science/article/pii/B0080430767006082.

[48] Sambit Mohapatra, Heinrich Gotzig, Senthil Kumar Yogamani, Stefan Milz, and Raoul Zöllner. Exploring deep spiking neural networks for automated driving applications. In Alain Trémeau, Giovanni Maria Farinella, and José Braz, editors, *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2019, Volume 5: VISAPP, Prague, Czech Republic, February 25-27, 2019*, pages 548–555. SciTePress, 2019. doi: 10.5220/0007469405480555.

[49] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[50] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015. ISSN 1662-453X. URL https://www.frontiersin.org/article/10.3389/fnins.2015.00437.

[51] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[52] Jose C. Principe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441915699, 9781441915696.

[53] Ignacio Reyes. Monitoreo y aprendizaje de redes neuronales utilizando medidas de información y su aplicación en detección de eventos astronómicos transitorios. *Uchile*, 2019. URL http://repositorio.uchile.cl/handle/2250/170542.

[54] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W Lindsay, Kenneth D Miller, Richard Naud, Christopher C Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P Kording. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, October 2019. ISSN 1097-6256. doi: 10.1038/s41593-019-0520-2.

[55] Gonzalo Sanchez, Murali Rao, and Jose Principe. Measures of entropy from data using infinitely divisible kernels. *IEEE Transactions on Information Theory*, 61(1):535–548, Jan 2015.

[56] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27 (3):379–423, 1948.

[57] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. doi: https://doi.org/10.1016/j.physd.2019.132306.

[58] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf.

[59] Hava T. Siegelmann. *Computation Beyond the Turing Limit*, pages 153–164. Birkhäuser Boston, Boston, MA, 1999. ISBN 978-1-4612-0707-8. URL https://doi.org/10.1007/978-1-4612-0707-8_12.

[60] Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39 – 47, 2019. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.11.014. URL http://www.sciencedirect.com/science/article/pii/S0925231218313420.

[61] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47 – 63, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.12.002.

[62] LibreTexts Team. Introduction to random variables, Jan 2020. URL https://stats.libretexts.org/@go/page/3258.

[63] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015.

[64] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Unkown*, pages 368–377, 1999.

[65] Jos Unpingco. *Python for Probability, Statistics, and Machine Learning*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319307150.

[66] Madhavun Vasu and Eduardo Izquierdo. Information bottleneck in control tasks with recurrent spiking neural networks. *CoRR*, abs/1706.01831, 2017. URL http://arxiv.org/abs/1706.01831.

[67] David Watson. Central limit theorem for the continuous uniform distribution, 2010. URL http://demonstrations.wolfram.com/CentralLimitTheoremForTheContinuousUniformDistribution. Last visited on 25/7/2021.

[68] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[69] Kristoffer Wickstrøm, Sigurd Løkse, Michael Kampffmeyer, Shujian Yu, Jose Principe, and Robert Jenssen. Information Plane Analysis of Deep Neural Networks via Matrix-Based Renyi's Entropy and Tensor Kernels, 2019.

[70] Bojian Yin, Federico Corradi, and Sander M. Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks. In *International Conference on Neuromorphic Systems 2020*, ICONS 2020, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388511. doi: 10.1145/3407197.3407225. URL https://doi.org/10.1145/3407197.3407225.

[71] Shujian Yu, Luis Gonzalo Sánchez Giraldo, Robert Jenssen, and José Príncipe. Multivariate extension of matrix-based rényiś α-order entropy functional. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(11):2960–2966, 2020.

[72] Xi Yu, Shujian Yu, and Jose C. Principe. Deep deterministic information bottleneck with matrix-based entropy functional, 2021.

[73] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. CENGAGE, 2020. https://d2l.ai.

[74] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgnXpVYwS.

[75] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgnXpVYwS.

[76] R.E. Ziemer and W.H. Tranter. *Principles of Communications: Systems, Modulation, and Noise*. Wiley, 2002. ISBN 9780471392538. URL https://books.google.cl/books?id=6ROfAQAAIAAJ.

Appendices

Appendix A

Linear Transformation Approximates a Normal Distribution

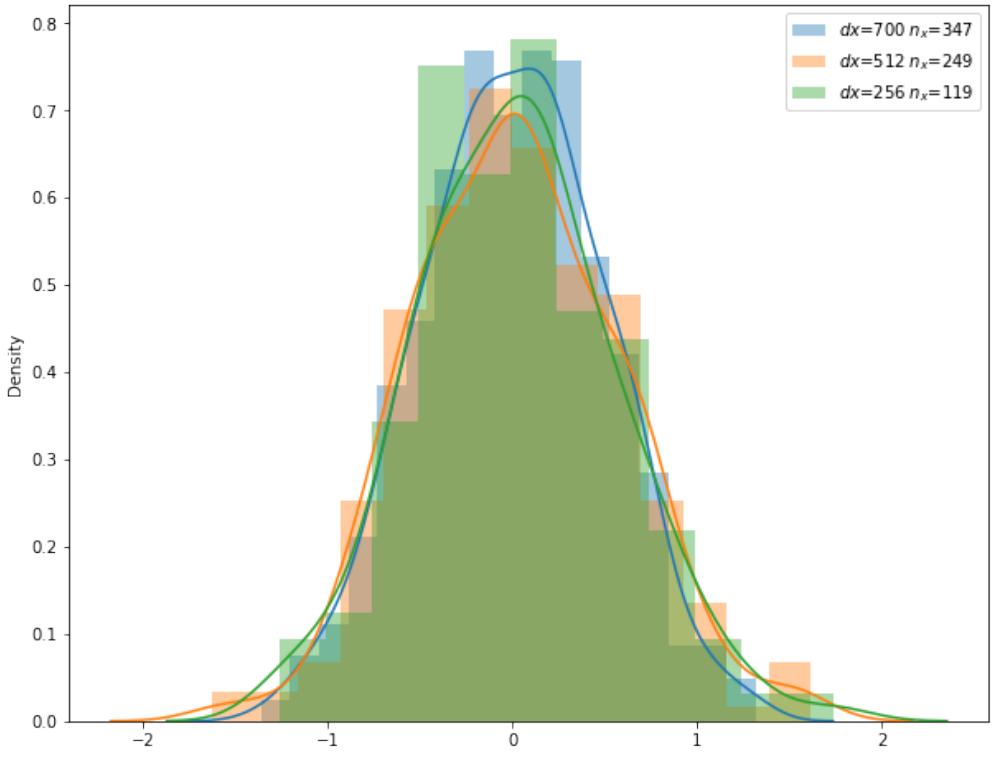
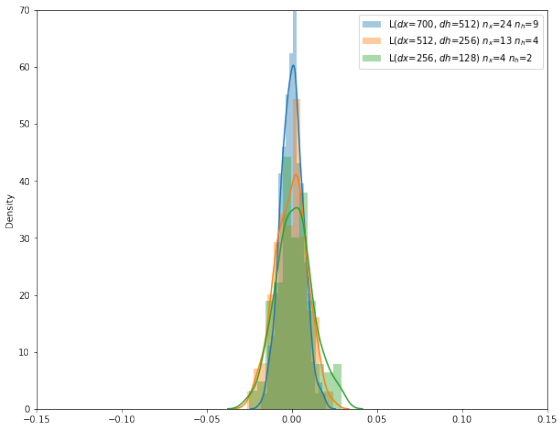


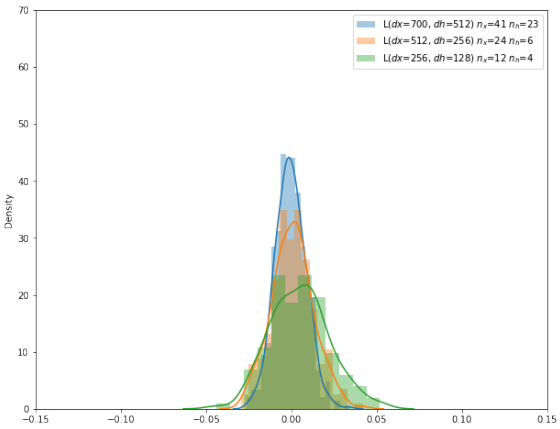
Figure A.1: Kernel density estimation of a linear transformation between a random binary vector and a uniform distributed matrix, where n_x is the number of samples taken from the uniform distributed matrix.

Appendix B

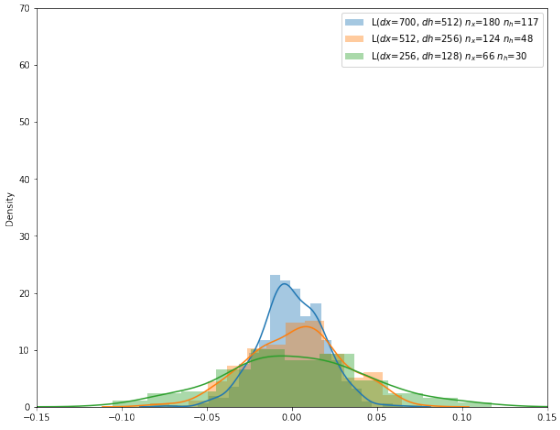
Input Current PDF Approximation



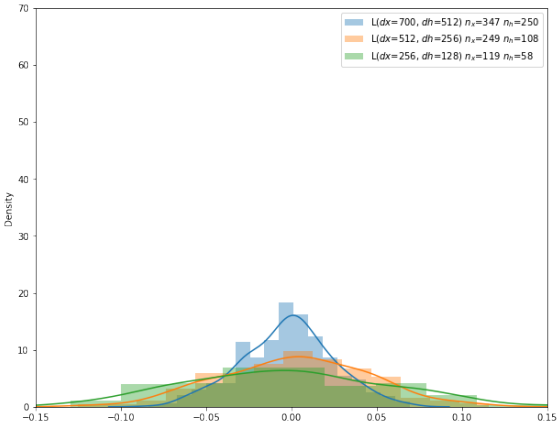
a) Approximation when $n_x = 24$, $n_h = 9$ in the first layer (color light blue)



b) Approximation when $n_x = 41$, $n_h = 23$ in the first layer (color light blue)



c) Approximation when $n_x = 180$, $n_h = 117$ in the first layer (color light blue)



d) Approximation when $n_x = 347$, $n_h = 250$ in the first layer (color light blue)

Figure B.1: Probability density approximation of the input current (Eq. 2.14) using CLT, when the ratio of sample and layer dimensions is tested with different values.

Appendix C

Maximum Scores of Baselines

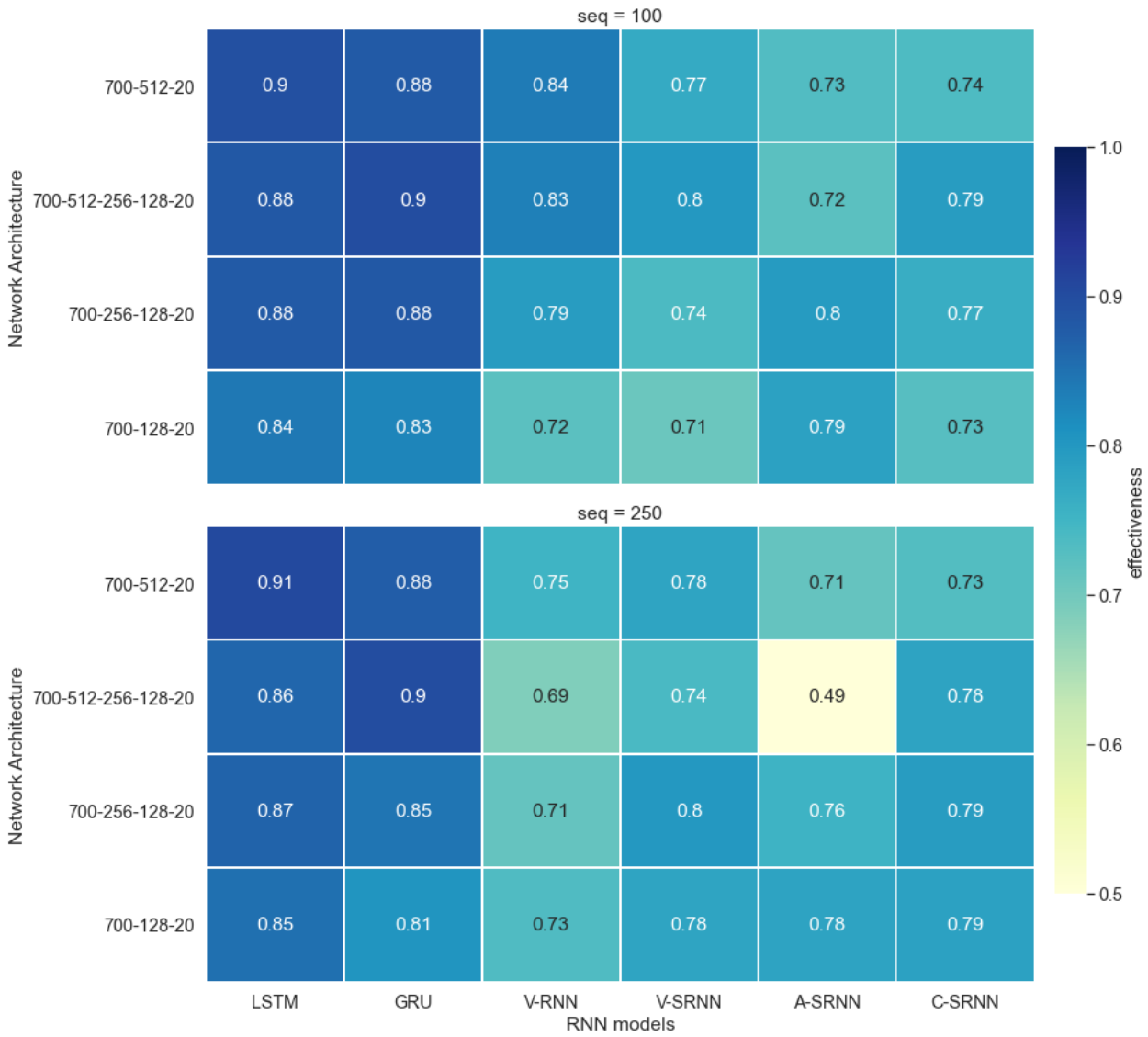


Figure C.1: Maximum accuracy of traditional RNNs and SRNNs for the SHD dataset.