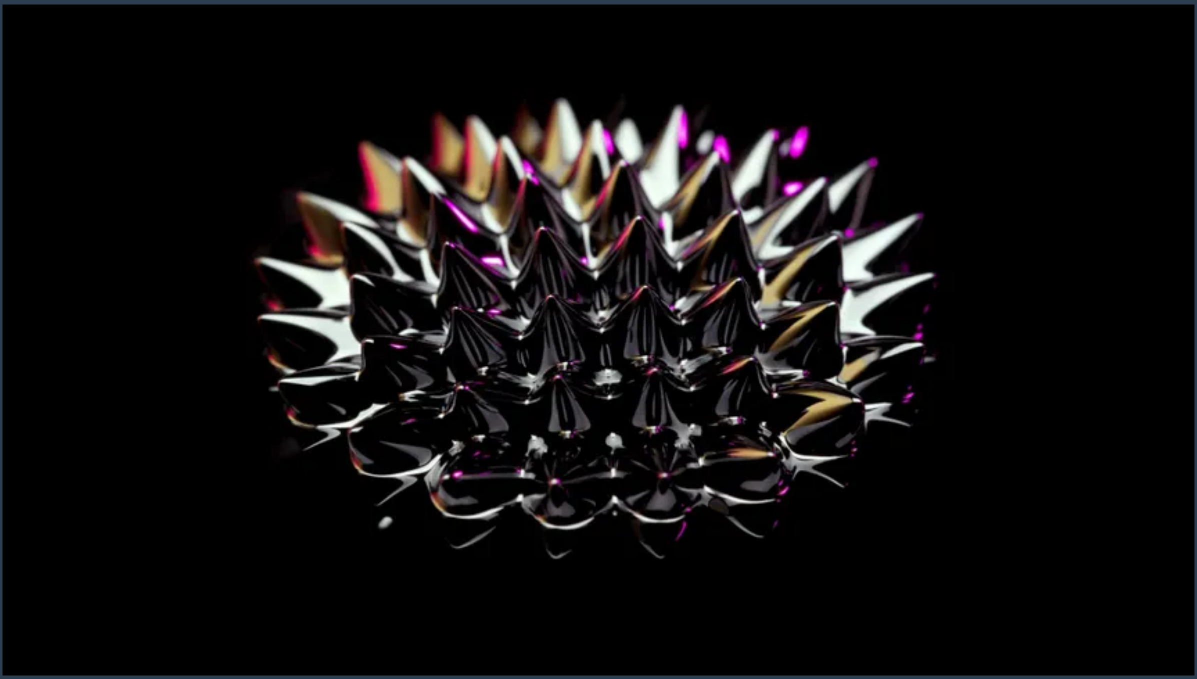
Final Report

Spiking Neural Networks
Sound Detection and Classification



COURREGE Téo
GANDEEL Lo'aï

Date: February 25, 2024

Contents

List of Figures

List of Tables

1 Introduction

In the last report, we explored the foundational aspects of Spiking Neural Networks (SNNs) and their potential applications, focusing on audio classification tasks. We delved into the theoretical underpinnings of SNNs, studying various neuron models and encoding methods such as rate, latency, and frequency coding. Additionally, we addressed challenges related to data preprocessing, verification, and the reconstruction of audio signals using Mel-frequency cepstral coefficients (MFCCs).

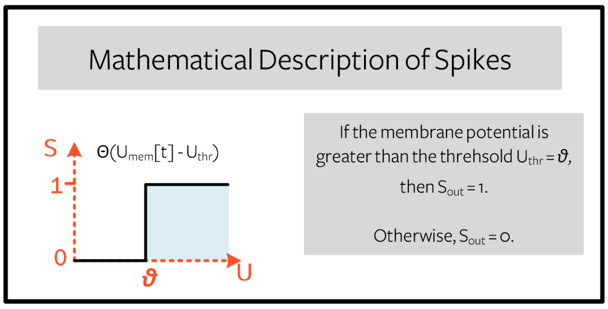

Mathematical Description of Spikes
S
1
0
$\Theta(U_{mem}[t] - U_{thr})$
ϑ
U
If the membrane potential is greater than the threhsold $U_{thr} = \vartheta$, then $S_{out} = 1$.
Otherwise, $S_{out} = 0$.

Figure 1: Mathematical description of a Spike[1]

Building upon this foundation, the current report will delve deeper into the intricacies of SNNs, specifically addressing the non-differentiability issue inherent in these networks. Non-differentiability poses a challenge in applying traditional gradient-based optimization techniques commonly used in training Artificial Neural Networks (ANNs). We will explore strategies to tackle this issue and optimize SNNs effectively.

Furthermore, we will introduce the concept of Convolutional Spiking Neural Networks (CSNNs), extending the discussion beyond simple SNN architectures. CSNNs leverage the spatial hierarchies present in convolutional neural networks (CNNs) and integrate them with the temporal dynamics of SNNs. This fusion holds promise for tasks like image recognition, where both spatial and temporal features play crucial roles.

In the practical implementation section, we will present a small program showcasing our results. This program will include the application of CSNNs in a specific task, demonstrating the capabilities and potential advantages of this hybrid architecture.

Throughout this report, our aim is to provide a comprehensive understanding of the advancements and challenges in the realm of Spiking Neural Networks, offering insights into their unique characteristics and applications.

2 Reminder on Spikes and Spiking Neural Networks

Through this section, we aim to provide a comprehensive and visual view of the concepts and mechanisms behind Spiking Neural Networks (SNNs). We will start by revisiting the fundamental aspects of spikes and their encoding, followed by an overview of neuron models (more specifically the Leaky Integrate-and-Fire (LIF) neuron model). We will then delve into the architecture of SNNs and their convolutional variant, Convolutional Spiking Neural Networks (CSNNs).

2.1 Spikes encoding

The idea behind spikes is simple: we need to encode some data (let's say an image) into some other kind of data that has a temporal dependency. To do so, we will need to define time steps, a number of steps, and a threshold. In a rate encoding scheme, we take the values of the pixels of the image and make them pass through a function that will output or not a spike (for example, a Bernoulli function).

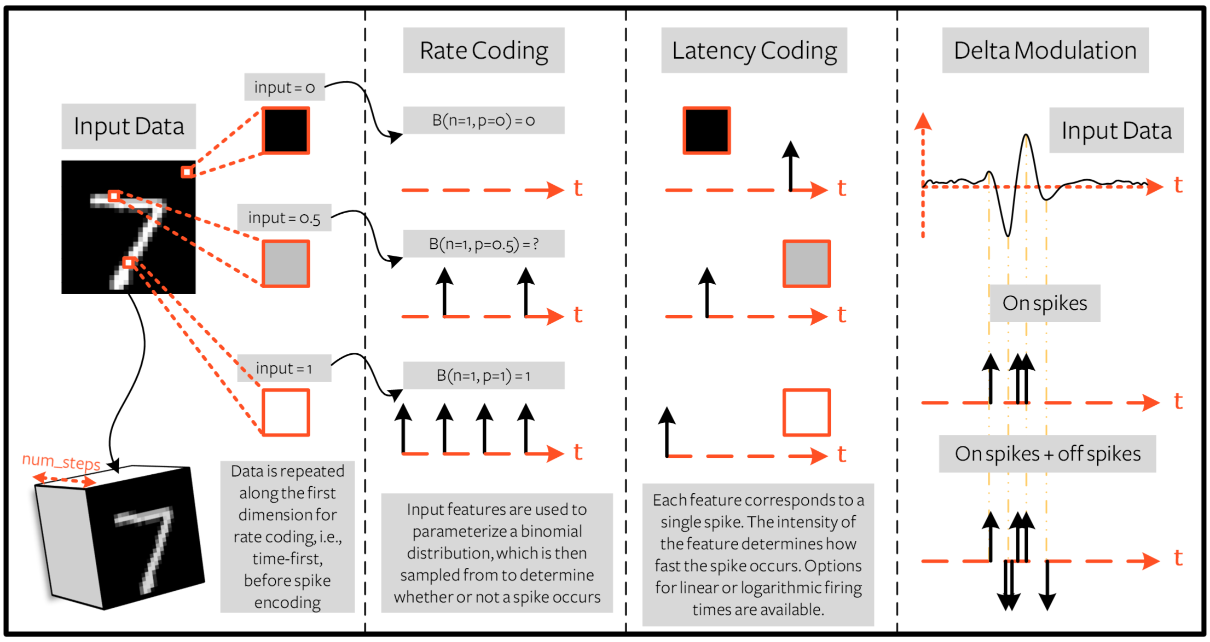

Input Data
input = 0
input = 0.5
input = 1
num_steps
Data is repeated along the first dimension for rate coding, i.e., time-first, before spike encoding
Rate Coding
B(n=1, p=0) = 0
B(n=1, p=0.5) = ?
B(n=1, p=1) = 1
t
Input features are used to parameterize a binomial distribution, which is then sampled from to determine whether or not a spike occurs
Latency Coding
Each feature corresponds to a single spike. The intensity of the feature determines how fast the spike occurs. Options for linear or logarithmic firing times are available.
Delta Modulation
Input Data
On spikes
On spikes + off spikes

Figure 2: Spike encodings of an image [1]
Source : https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_1.html

2.1.1 Rate encoding

Example

Let's say we have a white pixel (value close to 1) and a black pixel (value close to 0). We will then pass these values through a Bernoulli function with a threshold of 0.5, if the value is above 0.5, we will probably output a spike (1); if not, we will output 0.

This way, we end up with an approximation of the image in the form of spikes (binary values !). In order for it to be efficient, we will need encode the image multiple times. So, we will encode the image using the same method but with different time steps:

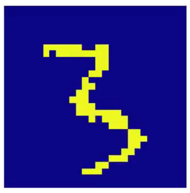


Figure 3: MNIST[1] $t = 0$

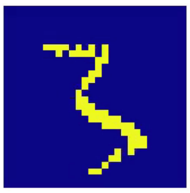


Figure 4: MNIST[1] $t = 15$

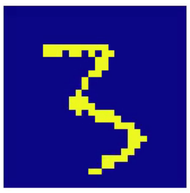


Figure 5: MNIST[1] $t = 32$

We would define a number of time steps n_{steps} and encode the image n_{steps} times. The more time steps we have, the more precise the encoding will be :

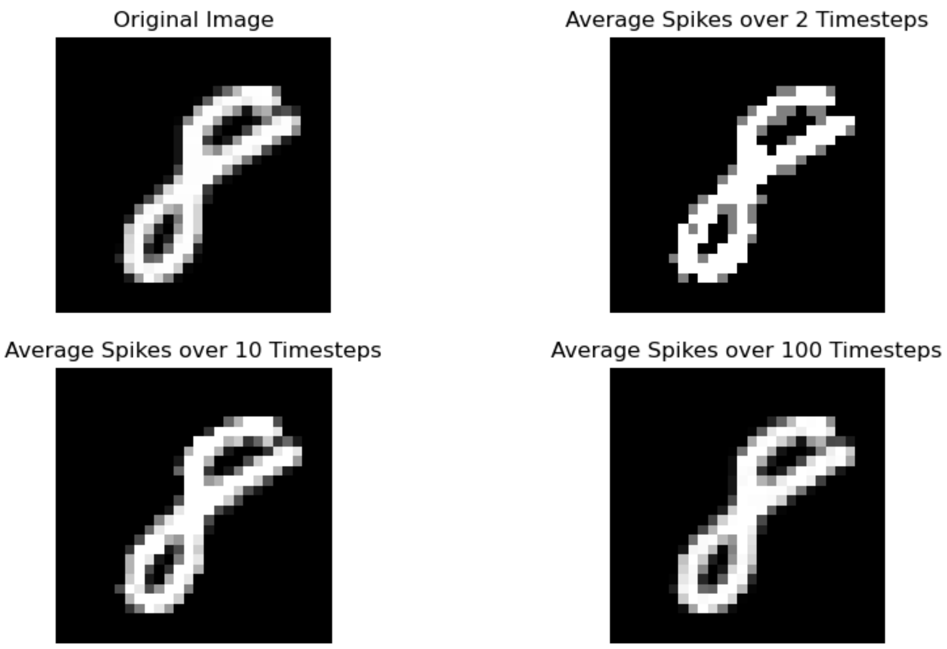

Original Image
Average Spikes over 2 Timesteps
Average Spikes over 10 Timesteps
Average Spikes over 100 Timesteps

Figure 6: Average of the encoding of the MNIST dataset

2.1.2 Latency encoding

2.1.3 Delta Modulation

2.2 Spiking neuron models

Until now, we have only talked about the encoding of the data. But another important aspect of Spiking neural network would concern the transmission of spikes and their associated information to the next layer.

We can basically distinguish three phases in a spiking neural network :

- Encoding the data into spikes
- Transmitting the spikes (and their associated information) to the next layer
- Decoding the spikes into a meaningful output

To do so, we would need a type of neuron that "fires" spikes when it receives enough input (whatever the inputs are). One might find different kinds of neurons, each with its own characteristics and properties.

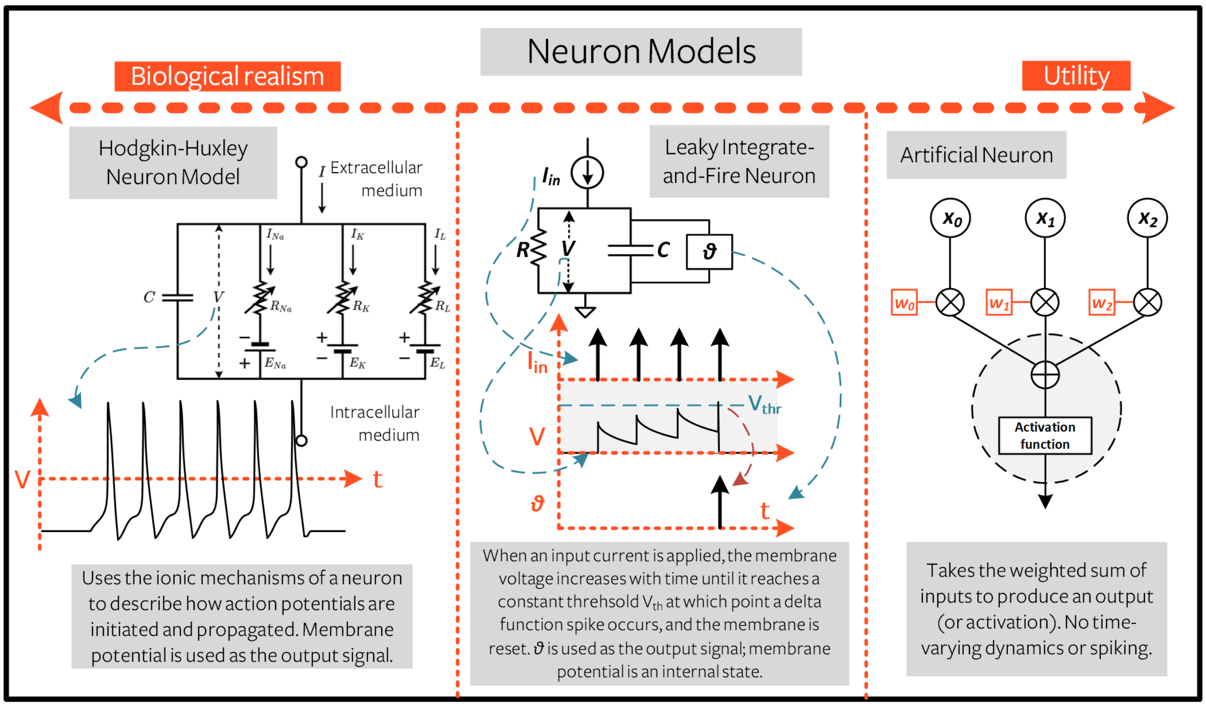

Neuron Models
Biological realism
Utility
Hodgkin-Huxley Neuron Model
Extracellular medium
Intracellular medium
Leaky Integrate-and-Fire Neuron
Artificial Neuron
Activation function
Uses the ionic mechanisms of a neuron to describe how action potentials are initiated and propagated. Membrane potential is used as the output signal.
When an input current is applied, the membrane voltage increases with time until it reaches a constant threhsold V_{th} at which point a delta function spike occurs, and the membrane is reset. ϑ is used as the output signal; membrane potential is an internal state.
Takes the weighted sum of inputs to produce an output (or activation). No time-varying dynamics or spiking.

Figure 7: Neuron models with schemes

Bien comme ref : https://neuronaldynamics.epfl.ch/online/

2.2.1 Comparison of the different neuron models

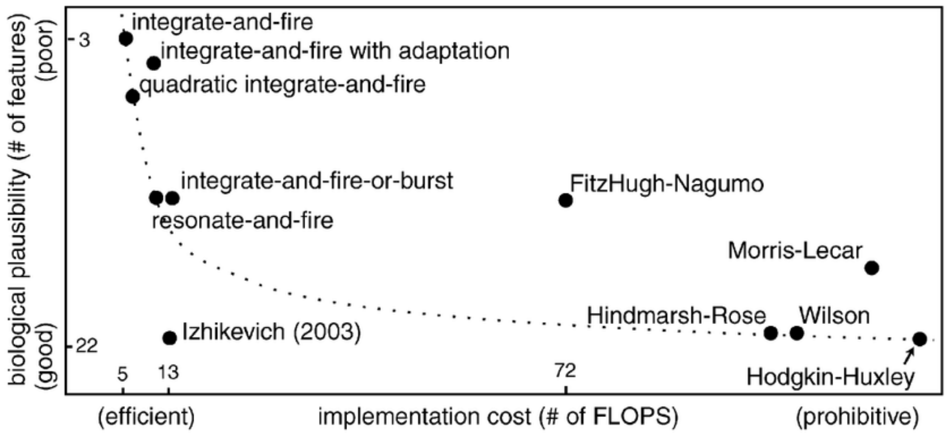


Figure 8: Comparison of the different neuron models
Source: https://www.researchgate.net/figure/Trade-off-between-biological-plausibility-with-respect-to-a-number-of-features-known-to_fig3_320409442

2.2.2 The choice of the Leaky Integrate-and-Fire Neuron Model

Initially, the LiF model was made as a simple model to mimic the behavior of a neuron.

> "In order to arrive at an equation that links the momentary voltage $u_i(t) - u_{rest}$ to the input current $I(t)$, we use elementary laws from the theory of electricity. A neuron is surrounded by a cell membrane, which is a rather good insulator. If a short current pulse $I(t)$ is injected into the neuron, the additional electrical charge $q = \int I(t')dt'$ has to go somewhere: it will charge the cell membrane. The cell membrane therefore acts like a capacitor of capacity C. Because the insulator is not perfect, the charge will, over time, slowly leak through the cell membrane. The cell membrane can therefore be characterized by a finite leak resistance R.
>
> The basic electrical circuit representing a leaky integrate-and-fire model consists of a capacitor C in parallel with a resistor R driven by a current $I(t)$ (cf the LiF "Neuron models with schemes")"[2]

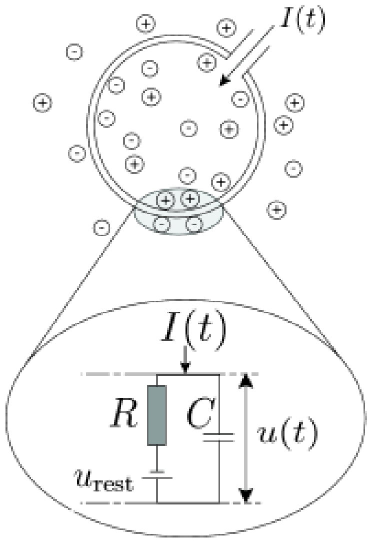


Figure 9: A neuron, which is enclosed by the cell membrane (big circle), receives a (positive) input current $I(t)$ which increases the electrical charge inside the cell. The cell membrane acts like a capacitor in parallel with a resistor which is in line with a battery of potential u_{rest}[2]

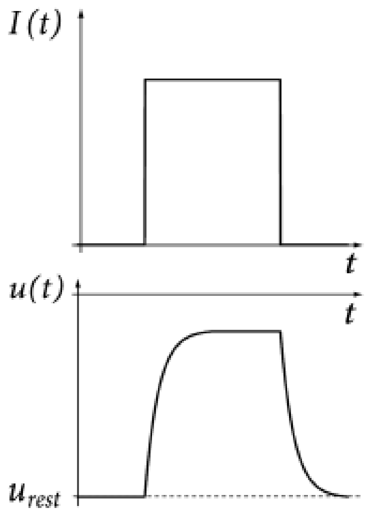


Figure 10: The cell membrane reacts to a step current (top) with a smooth voltage trace (bottom)[2]

2.2.3 Leaky Integrate-and-Fire Neuron Model

With the Leaky Integrate-and-Fire (LIF) model been the most used in practice. It can be seen as follows :

- We define a membrane potential U_m that will be updated at each time step

$$U_m[t+1] = \underbrace{\beta U_m[t]}_{\text{decay}} + \underbrace{WX[t+1]}_{\text{input}} - \underbrace{S[t]U_{\text{thr}}}_{\text{reset}} \quad [1] \tag{1}$$

- We define a threshold U_{thr} that will be used to reset the membrane potential when it is reached

- The output spike will be emitted following the following equation :

$$S(t) = \begin{cases} 1 & \text{if } U_m(t) \geq U_{\text{thr}} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

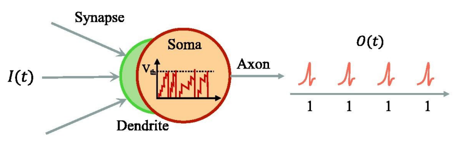


Figure 11: Neuron model basis[3]

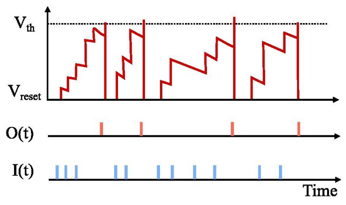


Figure 12: Generating spikes[3]

The algorithm defining the Lif neuron model in Snntorch have the following structure :

Algorithm 1 Snntorch : Leaky Integrate-and-Fire Neuron Model

```
Require: $input$, $mem_0$
 1: Initialize $beta$, $threshold$, $spike\_grad$, $surrogate\_disable$, $init\_hidden$, $inhibition$, $learn\_beta$,
    $learn\_threshold$, $reset\_mechanism$, $state\_quant$, $output$
 2: Initialize $Leaky.beta$ and $Leaky.threshold$ if they are learnable
 3: for each $input$ in $batch$ do
 4:     if $reset\_mechanism = "subtract"$ then
 5:     end if
 6:     Compute $U[t+1] = \beta U[t] + I_{in}[t+1] - RU_{thr}$ if $reset\_mechanism = "subtract"$
 7:     Compute $U[t+1] = \beta U[t] + I_{syn}[t+1] - R(\beta U[t] + I_{in}[t+1])$ if $reset\_mechanism = "zero"$
 8:     Compute $spk$ and $mem_1$ for each element in the batch
 9:     Return $spk$ and $mem_1$
10: end for
```

In pratcice, when using Snntorch we would create 2 arrays, one for the membrane potential and one for the spikes.

TODO

2.3 Spiking Neural Networks

2.4 Convolutional Spiking Neural Networks

3 SNN - Behind the scenes

3.1 How to train a SNN

3.2 Dead Neuron Problem

To train a spiking neural network, we aim to adjust the weights based on the loss gradient, minimizing the overall loss. Backpropagation achieves this through a chain of derivatives:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial S} \underbrace{\frac{\partial S}{\partial U}}_{\{0,\infty\}} \frac{\partial U}{\partial I} \frac{\partial I}{\partial W}$$

Here, $\mathcal{L}$ is the loss, W represents weights, S is the output, U is the activation function, and I is the input.

The challenge lies in the term $\frac{\partial S}{\partial U}$, which takes values between 0 and ∞. The derivative of the Heaviside step function from the input (U) is the Dirac Delta function. This function is 0 everywhere except at the threshold θ, where it tends to infinity. Consequently, the gradient is often nullified to zero (or saturated if θ precisely aligns with the threshold), hindering learning. This issue is commonly known as the **dead neuron problem**. There are multiple ways to address this issue.

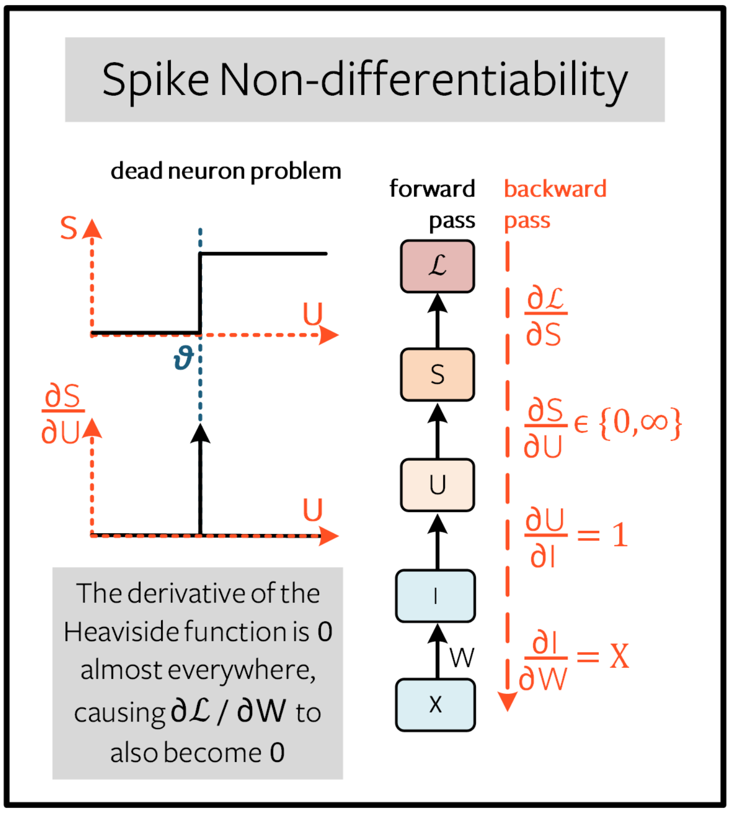

Spike Non-differentiability
dead neuron problem
S
U
ϑ
∂S/∂U
U
The derivative of the Heaviside function is 0 almost everywhere, causing ∂ℒ / ∂W to also become 0
forward pass
backward pass
ℒ
S
U
I
W
X
∂ℒ/∂S
∂S/∂U ϵ {0,∞}
∂U/∂I = 1
∂I/∂W = X

Figure 13: Dead neuron problem[1]

3.3 Shadow training

3.4 Surrogate Gradient

One way to address this non-differentiability issue would be to compute a gradient on a "relaxed" version of the non-differentiable function. This would mean approximating the Heaviside step function $S(U)$ with a differentiable function $f(U)$, such as the sigmoid function or the $\arctan$ function. For example:

3.4.1 Loss function treatment

In order for this to be taken into account in the loss function (and because we now have to take the time as a parameter), we have to perform an operation on the loss function for all its time steps.

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}[t]}{\partial W} = \sum_t \sum_{s \leq t} \frac{\partial \mathcal{L}[t]}{\partial W[s]} \frac{\partial W[s]}{\partial W}$$

4 Practical implementation

$$S(U) \approx \frac{1}{1+e^{-\beta(U-\theta)}}$$

Or alternatively

$$S(U) \approx \frac{1}{\beta} \arctan(\beta(U-\theta))$$

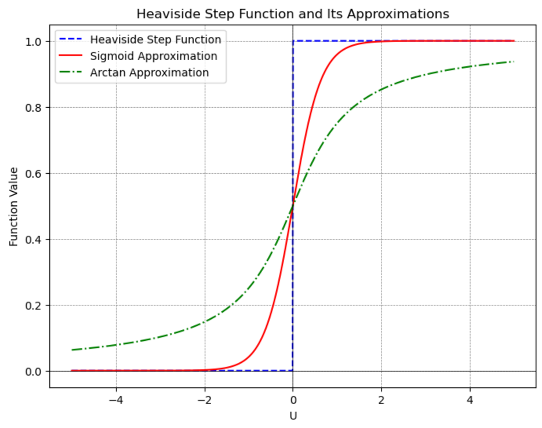


Figure 14: Visualization of the Heaviside step function approximation[3]

References

[1] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, *Training spiking neural networks using lessons from deep learning*, Proceedings of the IEEE, 111 (2023), pp. 1016–1054.

[2] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From single neurons to networks and models of cognition*, Cambridge University Press, 2014.

[3] X. Liao, Y. Wu, Z. Wang, D. Wang, and H. Zhang, *A convolutional spiking neural network with adaptive coding for motor imagery classification*, Neurocomputing, 549 (2023), p. 126470.