Convolutional and dynamical spintronic neural networks

Erwan Plouet

université PARIS-SACLAY

Convolutional and dynamical spintronic neural networks

Réseaux de neurones convolutifs et dynamiques basés sur des composants spintroniques

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 564, Physique en Île-de-France (PIF)
Spécialité de doctorat : Physique
Graduate School : Physique. Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **Laboratoire Albert Fert (Université Paris Saclay, CNRS, Thales)**, sous la direction de **Julie GROLLIER**, directrice de recherche, le co-encadrement de **Frank Alice MIZRAHI**, ingénieur

Thèse soutenue à Paris-Saclay, le 07 novembre 2024, par

Erwan PLOUET

Composition du jury

Membres du jury avec voix délibérative

Damien QUERLIOZ Directeur de recherche CNRS, Université Paris-Saclay	Président
Amalio FERNANDEZ PACHECO Professeur, Technische Universität Wien (Autriche)	Rapporteur & Examinateur
Joseph FRIEDMANN Professeur associé, University of Texas at Dallas (Etats-Unis)	Rapporteur & Examinateur
Jean Anne INCORVIA Professeur associé, University of Texas at Austin (Etats-Unis)	Examinatrice

THESE DE DOCTORAT

NNT : 2024UPASP120

université PARIS-SACLAY | ÉCOLE DOCTORALE Physique en Île-de-France (PIF)

Titre : Réseaux de neurones convolutifs et dynamiques basés sur des composants spintroniques
Mots clés : Spintronique, IA, Réseaux de neurones convolutifs, Réseaux de neurones dynamiques

Résumé : Cette thèse aborde le développement de composants spintroniques pour le calcul neuromorphique, une approche novatrice visant à réduire la consommation énergétique significative des applications d'intelligence artificielle (IA). L'adoption généralisée de l'IA, y compris des très grands modèles de langage tels que ChatGPT, a entraîné une augmentation des besoins énergétiques, les centres de données consommant environ 1 à 2 % de l'énergie mondiale, avec une projection de doublement d'ici 2030. Les architectures hardware traditionnelles, qui séparent la mémoire et les unités de traitement, ne sont pas adaptées aux tâches d'IA, car les réseaux de neurones nécessitent un accès fréquent à de nombreux paramètres stockés en mémoire, entraînant une dissipation excessive d'énergie. Le calcul neuromorphique, inspiré par le cerveau humain, fusionne les capacités de mémoire et de traitement dans un même dispositif, réduisant potentiellement la consommation d'énergie. La spintronique, qui manipule le spin des électrons plutôt que la charge, offre des composants capables de fonctionner à moindre puissance et de fournir des solutions de traitement efficaces.

Cette thèse est divisée en deux parties principales. La première partie se concentre sur la réalisation expérimentale d'un réseau de neurones convolutif hybride hardware-software (CNN) utilisant des composants spintroniques. Les synapses spintroniques, qui fonctionnent avec des signaux radiofréquences, permettent un multiplexage en fréquence pour réduire le besoin de nombreuses connexions physiques dans les réseaux de neurones. Ce travail de recherche explore divers designs de synapses basées sur des spin diodes AMR, chacune avec des spécificités différentes, et démontre l'intégration de ces synapses dans un CNN matériel. Une réalisation importante a été l'implémentation d'une couche convolutive spintronique au sein d'un CNN qui, combinée à une couche entièrement connectée en software, a réussi à classifier des images du dataset FashionMNIST avec une précision de 88 %, se rapprochant des performances d'un réseau purement software. Les principaux résultats incluent le développement et le contrôle précis des synapses spintroniques, la fabrication de chaînes synaptiques pour la somme pondérée dans les réseaux de neurones, et la mise en œuvre expérimentale réussie d'un CNN hybride avec des composants spintroniques sur une tâche complexe.

La deuxième partie de la thèse explore l'utilisation des nano-oscillateurs spintroniques (STNOs) pour traiter des signaux dépendants du temps à travers leurs dynamiques transitoires. Les STNOs présentent des comportements non linéaires qui peuvent être exploités pour des tâches complexes comme la classification de séries temporelles. Un réseau de STNOs simulés a été entraîné pour discriminer entre différents types de séries temporelles, démontrant des performances supérieures par rapport aux méthodes de calcul par réservoir standards. Nous avons également proposé et évalué une architecture de réseau multicouche de STNOs pour des tâches plus complexes, telles que la classification de chiffres manuscrits présentés pixel par pixel. Cette architecture a atteint une précision moyenne de 89,83 %, similaire à un réseau de neurones récurrents à temps continu (CTRNN) standard équivalent, indiquant le potentiel de ces réseaux à s'adapter à diverses tâches dynamiques. De plus, des méthodes ont été établies pour faire correspondre la dynamique des dispositifs avec les échelles de temps des entrées, cruciales pour optimiser les performances des réseaux de neurones dynamiques. Nous avons démontré qu'un réseau multicouche de STNOs couplés peut être efficacement entraîné via la rétropropagation de l'erreur dans le temps, soulignant l'efficacité et le passage à l'échelle possible du calcul neuromorphique spintronique.

Cette recherche a démontré que les réseaux spintroniques peuvent être utilisés pour mettre en œuvre des architectures spécifiques et résoudre des tâches complexes.

université PARIS-SACLAY | ÉCOLE DOCTORALE Physique en Île-de-France (PIF)

Title : Convolutional and dynamical spintronic neural networks
Keywords : Spintronic, AI, Convolutional neural networks, Dynamical neural networks

Abstract : This thesis addresses the development of spintronic components for neuromorphic computing, a novel approach aimed at reducing the significant energy consumption of AI applications. The widespread adoption of AI, including very large scale langage models like ChatGPT, has led to increased energy demands, with data centers consuming about 1-2% of global power, and projected to double by 2030. Traditional hardware architectures, which separate memory and processing units, are not well-suited for AI tasks, as neural networks require frequent access to large in-memory parameters, resulting in excessive energy dissipation. Neuromorphic computing, inspired by the human brain, merges memory and processing capabilities in the same device, potentially reducing energy use. Spintronics, which manipulates electron spin rather than charge, offers components that can operate at lower power and provide efficient processing solutions.

The thesis is divided into two main parts. The first part focuses on the experimental implementation of a hybrid hardware-software convolutional neural network (CNN) using spintronic components. Spintronic synapses, which operate with radio frequency signals, enable frequency multiplexing to reduce the need for numerous physical connections in neural networks. This research work explores various designs of AMR spin diode-based synapses, each with different specificities, and demonstrates the integration of these synapses into a hardware CNN. A significant achievement was the implementation of a spintronic convolutional layer within a CNN that, when combined with a software fully-connected layer, successfully classified images from the FashionMNIST dataset with an accuracy of 88%, closely matching the performance of the pure software equivalent network. Key findings include the development and precise control of spintronic synapses, the fabrication of synaptic chains for weighted summation in neural networks, and the successful implementation of a hybrid CNN with experimental spintronic components on a complex task.

The second part of the thesis explores the use of spintronic nano oscillators (STNOs) for processing time-dependent signals through their transient dynamics. STNOs exhibit nonlinear behaviors that can be utilized for complex tasks like time series classification. A network of simulated STNOs was trained to discriminate between different types of time series, demonstrating superior performance compared to standard reservoir computing methods. We also proposed and evaluated a multilayer network architecture of STNOs for more complex tasks, such as classifying handwritten digits presented pixel-by-pixel. This architecture achieved an average accuracy of 89.83% similar to an equivalent standard continuous time recurrent neural network (CTRNN), indicating the potential of these networks to adapt to various dynamic tasks. Additionally, guidelines were established for matching device dynamics with input timescales, crucial for optimizing performance in networks of dynamic neurons. We demonstrated that multilayer networks of coupled STNOs can be effectively trained via backpropagation through time, highlighting the efficiency and scalability of spintronic neuromorphic computing.

This research demonstrated that spintronic networks can be used to implement specific architectures and solve complex tasks. This paves the way for the creation of compact, low-power spintronic neural networks that could be an alternative to AI hardware, offering a sustainable solution to the growing energy demands of AI technologies.

Table des matières

Remerciements

Beaucoup de gens m'ont aidés et encouragé pendant cette thèse. J'aimerai remercier en tout premier lieu Estelle qui m'a soutenu indéfectiblement tout au long de ces trois ans, tu mérites autant que moi cette thèse. Je remercie également mes parents qui m'ont soutenu et écouté râler eux aussi quand j'en avais marre. Bien sûr je n'aurais pas pu faire tout ce travail sans mes encadrants Frank, Julie et Dedalo, qui m'ont accompagné tout au long de cette expérience et m'ont permis de découvrir tant d'aspects de la science. Un remerciement spécial à Dedalo qui a toujours été très patient et qui a toujours trouvé du temps pour discuter de tous les problèmes expérimentaux malgré toutes ses sollicitations extèrieures. Enfin merci à tout le laboratoire et sa bonne ambiance. Aux doctorants et post-docs de la team neuro avec qui j'ai beaucoup discuté, Dongshu, Arnaud, Théophile, Mohammed, Olivier, Katia qui devrait être membre honoraire en neuro, Naveen, Pankaj et les autres. Un grand merci aussi à tous ceux qui ont animé le groupe cantine, des doctorants des annés précédentes, Diane, Diana, Pauline, Yanis, Aya, à ceux qui sont maintenant au labo comme Sarah, William, Greeshmani, Malik, Nicolas,Meghan et tous les autres avec qui ont s'est bien amusés tous les midis. Merci à tous les autres que j'ai oublié de citer ici et qui m'ont accompagné pendant cette thèse.

Introduction

Nowadays, artificial intelligence (AI) is a booming field. Its applications are numerous : medical applications, autonomous driving, real-time prediction of trends in the economy and obviously natural language processing. The most famous representative of artificial intelligence models, the chatbot ChatGPT [1], is now present in the everyday lives of millions of people. However the increasing utilization of artificial intelligence raises a major concern about its energy consumption. Data centers where AI models are hosted and executed consume approximately 1 to 2% of the overall world power production and are estimated to double by 2030 [2]. This energy consumption needs to be addressed in order to fight climate change.

This energy consumption of data centers is mainly caused by the mismatch between traditional hardware architecture and AI models' requirements. Traditional hardware is based on an architecture where memory and processing units are separated, however AI neural networks require access to billions of in-memory parameters for processing. This creates a huge flow of data between memory and processor, responsible for most part of energy dissipation. To mitigate this cause of energy dissipation, a new approach for AI specialized hardware is developed : neuromorphic computing. Taking inspiration from the brain, the main idea is to implement these specialized hardware with nano-devices capable of merging memory and processing capacities. One of the fields that provides such nano-components is the field of spintronics, where instead of manipulating electrical charges, the spins of electrons become the main elements that encode and transmit information. This provides nano-components that can operate with lower power than standard electronics. Chapter 1 will display the state of the art of spintronic neuromorphic computing and other concurrent technologies.

In this thesis I will present the work I conducted to implement neural networks of different types with spintronic components, taking advantage of different properties offered by these components. This work is split into two parts, the experimental realization of a hybrid hardware-software convolutional neural network (CNN), and the simulation of a multilayer network of dynamical spintronic neurons.

Spintronic components can be addressed with radio frequency signals. This frequency connectivity can solve the major issue of wiring in hardware neural networks. Indeed, by implementing frequency multiplexing of signals combined with frequency selective devices, the number of required spatial

connections can be greatly reduced. The resulting network presents better compactness and simpler geometry. In the first part of the thesis, we demonstrate the controlled implementation of such frequency multiplexing through the realization of a spintronic convolutional layer. In chapter 2 the realisation of different designs of AMR spin diode-effect based synapses is demonstrated. These devices apply a tunable weight on radio-frequency inputs. They are made of a simple metallic magnetic layer conferring them robustness and relatively simple fabrication. In chapter 3 we demonstrate the concept of multiply-and-accumulate operation with these synapses connected in chains. We propose a mixed series and parallel configuration suited for future upscaling. We demonstrate precise control of the synaptic weights of a fabricated chain of synapses. Finally, in chapter 4, we introduce the experimental implementation of a convolutional neural network with a hardware spintronic convolutional layer and a software fully-connected layer. This network is trained taking into account the noise from the experimental setup and matches the expected accuracy on a clothing image classification task.

The use of the transient dynamics of physical devices can provide an energy-efficient way of processing time-varying signals. Spintronic nano oscillators (STNO) present non-linear dynamics that can be harnessed for such tasks. In the second part of this thesis I demonstrate that a network of simulated coupled spintronic oscillators with trainable transient dynamics can be used to classify different time series. In chapter 5 I demonstrate that a single-layer network of such oscillators can be trained via backpropagation through time to discriminate sine-square time series and show that its performance is superior to an equivalent network trained with a reservoir computing approach. In chapter 6 the architecture of a multilayer network of STNOs is proposed and evaluated on a more complex task, discriminating handwritten digits injected pixel by pixel. The impact of the matching between devices' characteristic relaxation time and input timescale is analyzed and guidelines for any network of dynamical neurons are proposed.

1 - State of the art

1.1 . Basic principles of neural networks

Standard logic-based algorithms struggle to solve cognitive tasks such as data classification, data prediction, or generation. Artificial neural networks have been developed to handle this specific kind of problems. An artificial neural network receives inputs and aims to produce an output corresponding to the task the network needs to solve. In the case of data classification, it receives the input data and outputs a label, in the form of a numerical value, to assign to this particular input.

Standard neural networks have been conceived by taking inspiration from the brain structure, more specifically from the sensory cortices. These areas consist of neurons connected through synapses, organized into successive layers. The first layer of neurons receives electrical inputs from the sensory organs of the body, neurons react non-linearly to these electrical signals and transmit them to the following layer of neurons through synapses. This hierarchical structure helps process complex sensory inputs and convert them into adequate signals sent to motor neurons.

While artificial neural networks (ANN) take inspiration from biological neural networks, there are some key differences. Biological neural networks present a much more complex structure and behavior than ANNs, with systematic dynamical behavior of neurons and synapses and complex connectivity. These dynamics are based on chemical processes and electrical spikes. Neurons and synapses can vary in type, and other cells are present in the brain and can contribute to cognitive tasks. The training mechanism is also different. In ANNs, the training is done to mathematically minimize an error function, whereas in biology the training is a complex, and not fully understood, mechanism. Finally, biological neural networks are very energy efficient, the human brain consumes around 20W, and training Chat GPT 3 took 1.287GWh [1].

1.1.1 . Artificial neurons and synapses

Artificial neural networks are composed of nodes called neurons in analogy with biology, these nodes are linked by connections called synapses. Similarly to a biological synapse, an artificial synapse transmits a signal from the output of a neuron to the input of another one, and a multiplicative weight is applied to the value of the signal. This weighting operation amplifies meaningful signals and discards information that is useless for the cognitive task. In

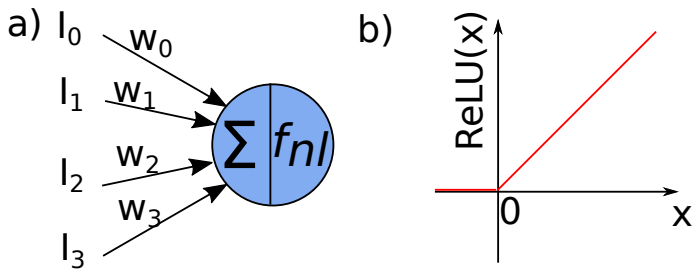


Figure 1.1 – a) A neuron performs a multiply and accumulate (MAC) operation on several inputs weighted by synapses before applying a non-linear activation function. b) Example of activation function : the Rectified Linear Unit.

general, the synaptic weight in artificial neural networks can be either positive or negative. An artificial neuron is a unit capable of summing several signals coming from different synapses, an operation called multiply and accumulate (MAC), and to apply a non-linear function to this sum, as illustrated in figure 1.1 a). The non-linear function is called the activation function. This node acts as a filter on the inputs that will permit or block the flow of complex information depending on the value of combined input signals. A common activation function is the Rectified Linear Unit function or ReLU [3], this function gives 0 when the input x is below 0 and x when x is above 0, as shown in figure 1.1 b). In the case of a neuron receiving inputs I_j from n synapses with weights w_j, the output of the neuron is $y = ReLU(\sum_{j=1}^{n} w_j I_j)$ which gives $y = 0$ if $\sum_{j=1}^{n} w_j I_j \leq 0$ and $y = \sum_{j=1}^{n} w_j I_j$ otherwise. In the first case, the information is not transmitted further in the network, and in the second case, this output will be weighted once again and sent to the next neurons. Additionally, a trainable bias b is generally present in the neuron's weighted sum so that the neuron receives : $\sum_{j=1}^{n} w_j I_j + b)$.

1.1.2 . Feed-forward fully-connected neural network

The most common artificial neural network architecture, inspired from the multilayer cortical structure, is called a feed-forward fully-connected neural network. Input signals are injected into the network by connecting them to the first layer of neurons via synapses. Then, the outputs of each neuron layer are propagated to the next layer of neurons as inputs via a new set of synapses. This structure is represented in figure 1.2 a). The network operation can also be viewed as a combination of successive mathematical transformations of the input, mixing the application of weight matrices representing the synaptic connectivity and non-linear multivariable functions corresponding to the neurons' activations. This multilayer architecture enables to discriminate non-linearly separable data points. The network projects the input data non-

linearly to a higher dimensional space where they can be linearly separated. The last layer of neurons will produce a final output that will be used for classification or data prediction. In the case of data prediction, the values of the neurons' outputs in the last layer directly correspond to the data being predicted. In the case of classification, the last layer needs to present as many neurons as the number of classes in the task. The neuron with the strongest output will define to which class an input signal does belong.

1.1.3 . Training

To solve a cognitive task correctly, the network needs to be trained on this specific task. This training consists of adjusting the network parameters, mainly the synaptic weights. To adjust these coefficients, we need to define a metric that evaluates the performance of the network on a task. This quantity is called the loss function and is designed to evaluate the difference between the current output result of the network for an input and the desired one, such as the number representing the class label for instance. This technique is called supervised learning and requires a large amount of pre-labelled inputs. Different types of loss functions exist and can be chosen depending on the nature of the task. For instance, the Mean Squared Error can be used for the prediction of data points, and the Cross Entropy Loss for a multi-class discrimination task. The most common and powerful training algorithm is called the backpropagation of errors. Once the loss function is evaluated for a given input, the derivatives of this error with respect to the synaptic weights and biases are calculated to quantify the impact of each trainable parameter on the error. To compute these derivatives the chain rule is applied successively through the layers of the network. For instance for a synaptic weight w_{i-1} from layer $i-1$, the derivative can be computed as follows : $\frac{\partial E}{\partial w_{i-1}} = \frac{\partial E}{\partial w_i}\frac{\partial w_i}{\partial w_{i-1}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_i}\frac{\partial w_i}{\partial y_{i-1}}\frac{\partial y_{i-1}}{\partial w_{i-1}}$, where y_i is the output of a neuron from layer i. This procedure is illustrated in figure 1.2 b). Once all the gradients have been computed, the trainable parameters are updated by subtracting a fraction of the gradients from their current value. For a weight w, the new value is $w - lr \times \frac{\partial E}{\partial w}$. This operation, called stochastic gradient descent [4] is designed to minimize the loss function. It requires that the hyperparameter lr, called the learning rate, is small compared to the trainable parameters values. Moreover it needs to be applied iteratively to find a minimum of the loss. The reached minimum is not guaranteed to be a global minimum. Improved versions of stochastic gradient descent have been developed to aim for more global minima, but they can still struggle if the landscape of the loss versus the parameters presents abrupt variations.

In order to train a network correctly, a large number of different examples

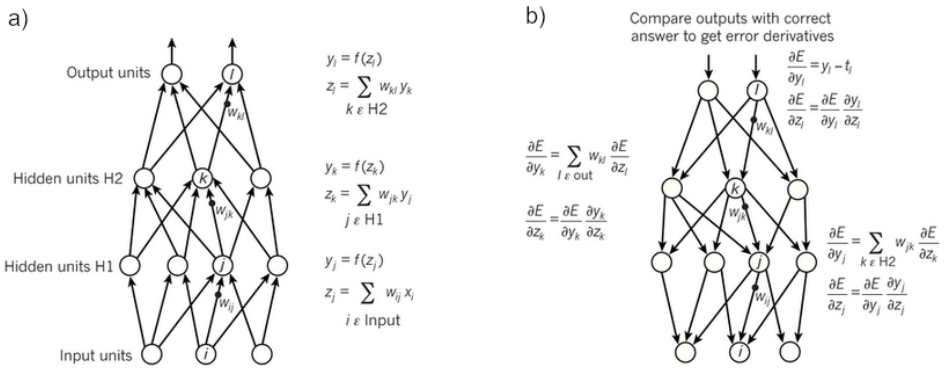


Figure 1.2 – a) Forward pass of a feed-forward network, inputs are injected into the first layer of neurons via synapses, and these neurons apply non linear transformations and transmit their outputs to the next layer of neurons via a new layer of synapses. b) Backward pass of a feed-forward network, the error is computed and the derivative of the loss with respect to each trainable parameter is computed via the chain rule. Weights are updated by subtracting a part of this derivative to minimize the error function. Image extracted from [5].

of inputs and their labels must be provided in the training phase. To implement the training, the dataset is split into at least two parts : one part will be used to train the network and another to test it. They are respectively called the training and testing sets. Generally, several inputs are processed in parallel and their errors are averaged before performing backpropagation, a method called batching. The training set is run through the network several times, each run being called an epoch. The number of epochs can vary from 10 to a few hundred. After training, the test set is run through the model to check if it is able to classify previously unseen data and thus generalize beyond the training examples.

1.1.4 . Convolutional neural network

Convolutional neural networks (CNNs) are artificial neural networks that have been specifically designed for signal processing, particularly for image recognition [6]. They are now the state of the art for various image-related tasks such as image classification and adversarial image generation. These networks have the ability to extract local features from input signals, enabling them to isolate meaningful patterns in images. For instance, they can be trained to recognize objects while discarding the background on which the objects are displayed. CNNs are resilient to noise and spatial transformations such as global image rotations and translations due to their local processing of information.

These networks present a feed-forward multilayer structure similar to standard fully connected networks. However, for each synaptic layer, the weights are designed to implement a convolution operation with multiple filters performed on the input image. This specificity makes CNNs intrinsically sparse in connections. Before the development of CNNs, convolutional filters were already widely used to extract local features, and many filters were conceived to extract relevant features in signals. However, in the case of CNNs, filters are made of trainable coefficients; consequently, they can learn to extract the most relevant features for a task without prior knowledge. The full convolution operation is performed on an image by sliding the convolutional filter over the input image and performing a MAC operation on the reduced area covered by the filter, which produces an output image called a feature map. This can be formulated mathematically as follows :

$$z_{h,g} = \sum_{i=1}^{K} \sum_{j=1}^{K} w_{i,j} x_{h+i,g+j} \tag{1.1}$$

Where $z_{h,g}$ is the value of the feature map at coordinates h along the height and g along the width, $x_{h+i,g+j}$ is the value of the input image at coordinates $h+i$ along the height and $g+j$ along the width, and K is the filter or kernel size. This convolution operation is represented in figure 1.3. In general, a CNN has several convolutional filters applied on the same input image; these filters each produce a feature map, and this additional dimension is called the channel dimension. The complete set of feature maps can thus be expressed as :

$$z_{h,g,c} = \sum_{i=1}^{K} \sum_{j=1}^{K} w_{i,j,c} x_{h+i,g+j,c} \tag{1.2}$$

Where N_c is the number of channels. Channels can, for instance, correspond to the RGB channels of an image.

Finally, to transform an image into a classification output, the size of the feature maps is reduced across the network, and the channel depth is increased. The size reduction is performed via an operation called max pooling; the image is divided into smaller squares, and in each square, only the highest value is kept. This ensures the transmission of the most important information and increases noise resilience. The final classification step is performed with a standard fully connected layer on the flattened output of the last layer, as represented in figure 1.4. The training and testing processes are similar to those of standard feed-forward networks.

1.2 . The different types of hardware neural networks

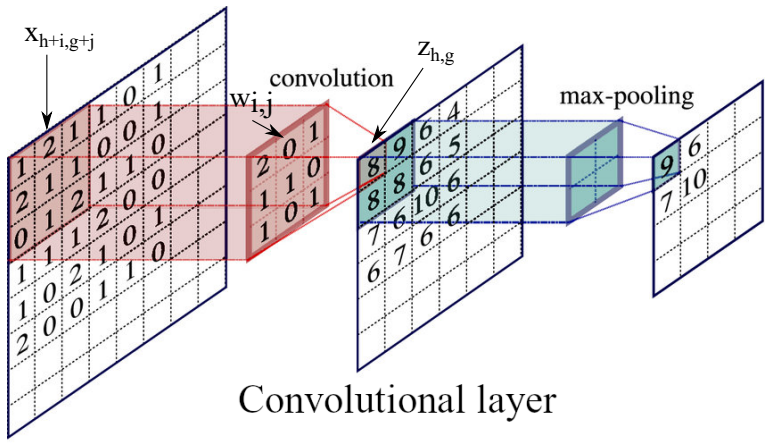


Figure 1.3 – Convolutional layer composed of a 3x3 pixel kernel sliding over the input image. A max pooling operation is performed on this output image with a kernel size of 2x2 pixels. Image from [7].

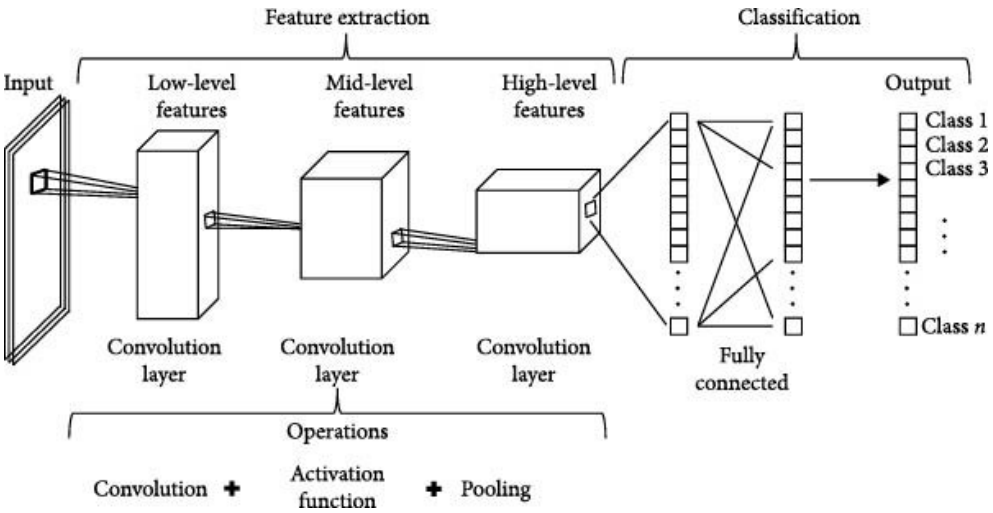


Figure 1.4 – Architecture of a convolutional neural network. The input image is processed by convolutional layers that extract increasingly refined local features, thus reducing its width and increasing its depth. The classification is performed by a fully connected layer that processes all output features. Image extracted from [8].

Artificial intelligence models are growing in size and complexity, with now billions of synapses (175 billion for GPT-3 [9]). These networks are challenging to implement even with conventional hardware and have significant energy consumption. As mentioned before, training GPT-3 consumed 1.287 GWh. To try to lower this energy consumption, new hardware is being developed to solve the limitations inherent in standard hardware. In this section, we will describe hardware designed for static inputs and that uses devices in their steady states; a second section will be devoted to networks using controllable transient dynamics for computation. Here, we will put a specific focus on hardware implementing CNNs, as this is the type of network that has been realized and will be presented in this thesis in chapters 2 to 4.

1.2.1 . Conventional hardware

Artificial intelligence has been theorized and developed since the 50's [10] [11] [12]. However, the field has experienced major growth over the last ten years. This boom is partly due to the emergence of new hardware, including the graphic processing unit and tensor processing unit (GPU, TPU), which facilitate the implementation of large networks and greatly reduce their training time. The GPU and TPU try to address the main limitation common to all standard computing architectures : the Von Neumann bottleneck [13]. Due to the separation between memory and processing units, data need to be transferred back and forth from memory to processors. This huge data flow is the main limitation in terms of energy consumption and speed for artificial intelligence tasks. GPUs and TPUs have been designed to reduce the impact of this bottleneck; they are made of parallelized cells comprising memory elements and cores for processing. This architecture minimizes the time and energy consumption of data exchange and allows the processing of large amounts of data in parallel. However, the memory and processing units remain separated, so the Von Neumann bottleneck remains an issue. This is visible in terms of energy consumption : to train a neural network to solve a natural language processing task on standard supercomputers consumes 1000 kWh, which is equivalent to 6 years of the human brain's energy consumption [14]. Other approaches try to bring memory and processors closer or to have processing units with integrated memory capacities. This approach, closer to the brain's nature, is called neuromorphic computing.

1.2.2 . Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) are chips composed of tens of thousands to a few million programmable logic blocks, integrated functions, and memory blocks [15][16]. These arrays of logic blocks can be freely connected to design custom architectures and are used to implement neural network accelerators. The main challenge of implementing neural networks on this type of hardware is mapping the network structure to the hardware's avai-

lable connectivity and computing power. FPGAs are especially interesting for CNN implementation due to the specific sparse structure of these networks [17][18]. Thanks to their high programmability, FPGAs can implement the sparsity and irregular parallelism of convolutions while using most of the available logic blocks, unlike GPUs that are poorly suited to exploit sparsity. Additionally, FPGA CNNs have part of their memory on chip, minimizing data transfers and thus reducing energy consumption compared to GPUs. CNNs solving the ImageNet [19] dataset have been implemented successfully [20][21], proving that this technology is compatible with large models and complex tasks.

Even if FPGAs start to compete with GPU implementations, they present certain limitations. The embedded memory is limited, which implies that most neural networks deployed on FPGAs are for now trained off-line, on GPUs and TPUs, and not directly in the hardware. The throughput of operations is smaller than that of the most advanced GPUs, reaching only 10 TFLOPs at maximum [22] compared to a few hundred [23]. Moreover, the implementation of the network architecture in hardware is a complex task and requires careful attention to ensure maximal performance. Finally, FPGAs require off-chip memory, which is another drawback that can slow down computation. This last limitation, common to all standard complementary metal-oxide-semiconductor (CMOS) hardware, could be tackled by a more neuromorphic approach with unconventional computing components that integrate memory.

1.2.3 . Photonic Neural Networks

Photonics is a promising field for neuromorphic computing. Computing with light offers several characteristics of photons that could speed up and reduce energy consumption in neural networks [24][25]. Light has a large bandwidth in wavelength that could help implement massive parallelism ; the very low intrinsic crosstalk between frequencies is also a key feature enabling this. The parallelisation can be further increased by adding channels of different light polarizations and orbital momenta. Additionally, light can be transmitted with low losses through optical fibers and waveguides, and manipulating the phase of beams rather than their intensity can help limit energy consumption.

Different implementations of neural networks have been achieved using photonic technologies, more specifically by implementing the synaptic connectivity and multiply-and-accumulate operations. We can divide them into three main categories : networks based on Mach-Zender interferometers, networks based on wavelength division multiplexing and composed of optical resonators, and a more general type of networks using spatially spread optical synapses of various types.

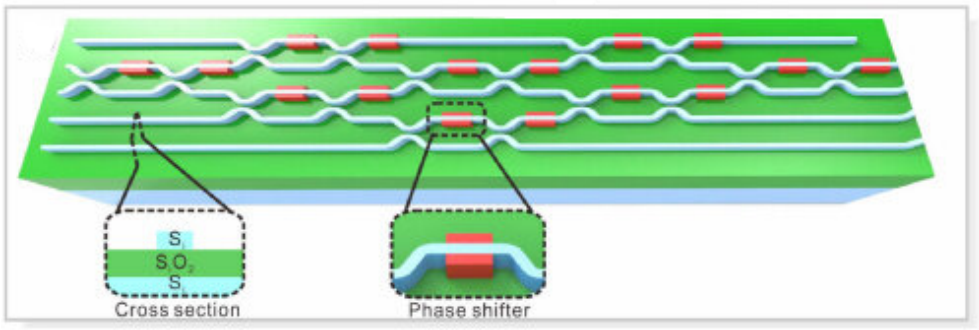


Figure 1.5 – A Mach-Zender interferometer mesh can implement a matrix multiplication on the amplitude of multiple coherent light beams. The coefficients of this matrix are implemented via phase shifters. Image from [26].

Mach-Zender Interferometer (MZI)-based neural networks [26] are designed to perform transformations on input vectors in the form of coherent light beams, each injected into a different waveguide. Each input value is represented by the light intensity of the associated beam. A mesh of MZI units is designed to implement a matrix multiplication applied to this input vector (see figure 1.5). The MZIs each have two tunable phase shifters and optical attenuators; they can be microfabricated and integrated. These phase shifters control the interference output of the MZI. They do not directly correspond to synaptic weight values; however, they are the trainable parameters that enable the implementation of the desired matrix multiplication. The weight tuning is achieved by changing the phase shifts by adjusting their length or refractive indices. This can be done mechanically, thermally by injecting carriers, or optoelectronically, either in a volatile or non-volatile manner. The main constraint of this architecture is the use of time-coherent light sources as inputs, which require precise and complex experimental conditions.

Wavelength Division Multiplexing (WDM) neural networks [27] have a structure closer to the classical architecture of a neural network, with the specificity of using the light wavelength dimension as connectivity. Each value of the input vector is represented by the light intensity of a light beam at a unique frequency. These inputs are weighted via optical resonators, typically microring resonators (MRRs) [28]. If the wavelength of an input signal matches the resonance frequency of the microring, part of this input signal will be transferred to the ring, thus diminishing the signal going through the input waveguide. By tuning the matching between the ring resonance frequency and the input frequency, the amount of transferred signal is controlled, which implements a synaptic weight. In a network design, the input waveguide is common to several resonators; its multiplexed output contains all the weighted signals to perform a MAC operation. A drop waveguide is added to discard the resonant

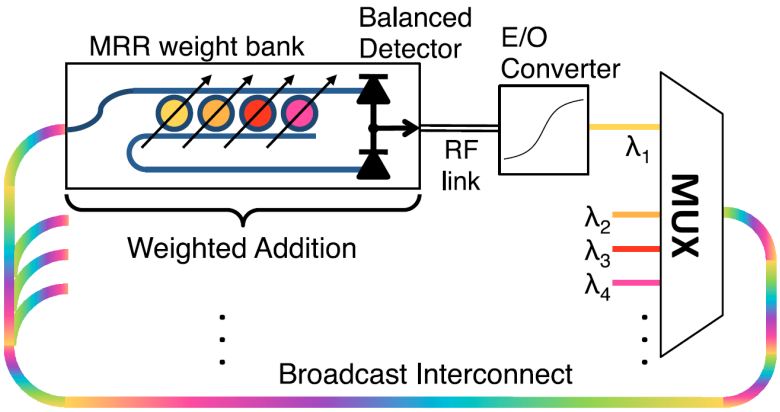


Figure 1.6 – Implementation of a neuromorphic architecture with optical microring resonators (MRR) as synapses. Beams with different wavelengths are sent to a set of MRR, each addressing only one input. Depending on the matching between input and resonance wavelength, a variable part of the input is transferred. A photodiode sums all transmitted weighted signals, and an electronic-to-optical component such as a laser can implement a non-linear function, thus forming a neuron. This architecture implements a recurrent loop to process temporal signals. Image from [30].

light that enters the resonator. The weight tuning is achieved by changing the length of the microring or its refractive index. This can be done mechanically, thermally by injecting carriers, or optoelectronically, either in a volatile or non-volatile manner. The described architecture is displayed in figure 1.6.

This network structure is compatible with CMOS components, integrable on-chip and scalable, unlike optical networks using macro-sized lenses. A challenge is the micro-resonator size, typically $25 \times 25\mu m$[24] and at best around $1 \times 1\mu m$ [14]. An architecture of this type adapted for convolutional neural networks has already been proposed for future implementation [29]. We will see in th next sections that similar principles can be applied to spintronic networks, with a much smaller footprint

Similar networks can be implemented with phase change materials (PCM). These materials can be switched optically or electrically between two states : amorphous or crystalline. The two phases generally exhibit different optical indices. This type of component has been used to implement optical memories with $Ge_2Sb_2Te_5$ as PCM [31]. An optical signal can be sent through a waveguide with such material on top, and the output will vary depending on the state of the PCM material. A multi-level memory has also been realized with several PCM islands that can be switched gradually; this memory implements a non-binary synaptic behavior [32]. A convolutional network has been reali-

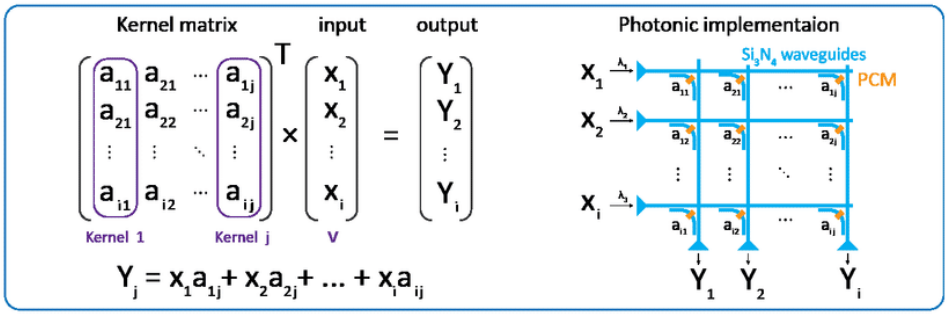


Figure 1.7 – Implementation of an optical matrix multiplication with PCM material as synapses organized in a crossbar array geometry. Image from [33].

zed with similar technology in a crossbar geometry [33] as shown in figure 1.7. Each input value is encoded as the amplitude of a light beam of a specific wavelength and injected through a line of the array. Along this line, PCM elements act as synapses and transmit part of the input light to a column depending on their phase state. The output of a column is a signal made of different weighted inputs that will be summed via a photodetector. This computation can be performed in parallel on multiple inputs if they are multiplexed in frequency. This network achieved 95.1% accuracy on a handwritten digits recognition task [34] with a high throughput of 2 tera MAC operations per second, which is comparable with CMOS performance, and low energy consumption of 17 fJ per MAC operation. This would lead, in the best case, to an efficiency of 7.0 TOPS per W, compared to the 4 TOPS per W of a standard graphics card [23].

Other photonic neural networks can be realized with spatial addressing of synapses. The values of the inputs can be implemented through their phase, amplitude, or both. The general principle is to shine an input vector as an optical image on an array of spatially spread optical elements that act on the phase or amplitude of the beams. The addressing of inputs by these synaptic elements can be done by using cylindrical lenses to direct each input beam onto a single column of the synapse array. The summation of the MAC operation is done with a similar lens [35]. Another technique is to use coherent light and let it diffract on an input plane encoding the input value ; this diffraction generates secondary waves for each input value that travel to address all the elements of an array made of optical synapses. This architecture can be reproduced in chained layers and is called a deep diffractive neural network [36].

The implementation of synaptic weights can be achieved with different elements that act on the amplitude and phase. The three main elements are

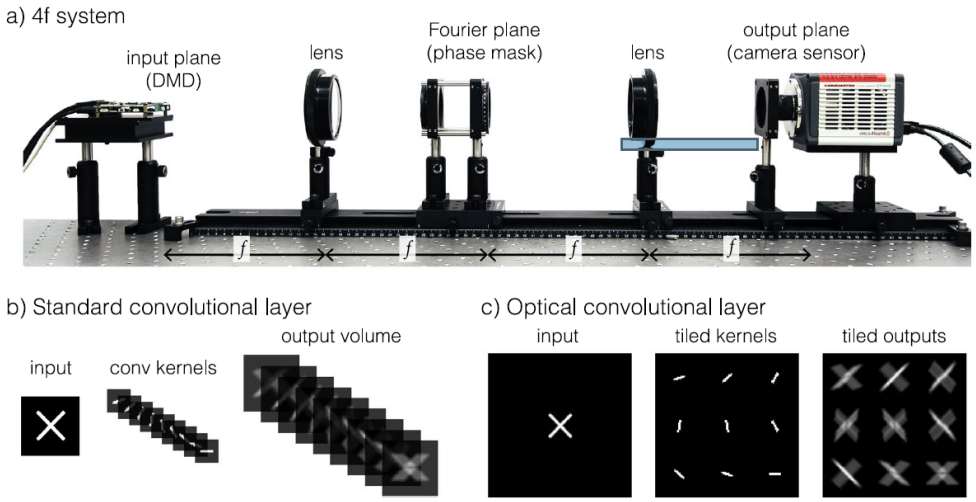


Figure 1.8 – Implementation of an optical convolutional neural network. The convolution is performed by placing a phase mask in the Fourier plane of the input image, thus applying the weights on the spatial Fourier transform of the image. This operation is equivalent to a convolution once another Fourier transform is performed. Image from [37].

spatial light modulators (SLM), diffractive optical elements (DOE), and holographic planes. SLMs have the specificity of being active elements that can be reconfigured, while DOEs are passive and cannot be modified after fabrication. An array of DOEs is also often called a phase mask. A holographic plane has intermediate characteristics; it's a medium with a writable optical index. By exposing the medium to a given wavelength, the optical index can be locally modified to store a synaptic weight. Some media are erasable and can thus be reconfigured.

Optics can also be advantageous for CNNs, as Fourier transforms and convolutions are closely related; a spatial convolution becomes a multiplication in the spatial Fourier space. A CNN implementation has been demonstrated with an array of DOEs [37], as displayed in figure 1.8. By placing this synaptic array in the Fourier plane of a lens placed after the input image, this synaptic array implements a multiplication of the Fourier transform of the input image, thus performing a convolution of the real image. The Fourier transform is inverted with a new lens to recover the convoluted image. This approach is very elegant and exploits the properties of Fourier optics; however, it requires a setup length of four times the focal distance of the lenses and is hard to scale or integrate.

As we have described in the preceding paragraphs, optics offers a wide variety of architectures to implement neural networks. Optical devices have

numerous advantages, such as high parallelism and low energy consumption ; however, they are generally hard to scale and integrate, with the notable exception of technologies using microring resonators. But even with MRR, the element size can be a limiting factor. Finally, neurons generally need to be implemented in CMOS, which can also limit the advantages brought by optics, such as speed and low power consumption.

1.2.4 . Memristor Neural Networks

Another promising technology for neuromorphic computing is the memristor. A memristor is a component capable of storing information in its resistance state. A memristor generally has several resistance levels and can be switched from one to another by applying current pulses. There are two main types of memristors : phase change memristors (PCM) [38], where the resistance depends on the crystalline arrangement of the material—highly conductive if the phase is crystalline and less when the phase is amorphous ; and filament-based memristors [39], where a conductive filament can be created or discarded in an insulator. Other types of memristors exist using various physical effects such as ferroelectricity [40], spintronic effects [41], or chemical redox reactions for polymer memristors [42].

Memristors can be wired together in a crossbar geometry to implement a matrix multiplication [43]. This is a key feature for implementing a synaptic layer with memristors. This matrix multiplication is simply achieved through Kirchoff's and Ohm's laws applied to a crossbar geometry where memristors are placed at each intersection of vertical and horizontal lines (see figure 1.9). The inputs are voltages applied to the horizontal metallic lines, and the outputs are the currents collected at the end of the vertical metallic lines. If each of the N memristors on row i and column j has a resistance $R_{i,j}$, the output current on column j is :

$$I_j = \sum_{i=0}^{N} \frac{V_i}{R_{i,j}} \tag{1.3}$$

Consequently, the conductances $G_{i,j} = \frac{1}{R_{i,j}}$ can implement synaptic weights and can be trained to solve a machine learning task. It is important to note that two memristors are required to implement the positive and negative possible values of a weight. These arrays of memristors have been used to build multilayer neural networks combined with CMOS neurons [44] or more complex memristor-based neurons [45].

Memristors present several advantages compared to standard CMOS computation components and to other neuromorphic hardware. They are fully CMOS compatible because they use simple physical quantities such as vol-

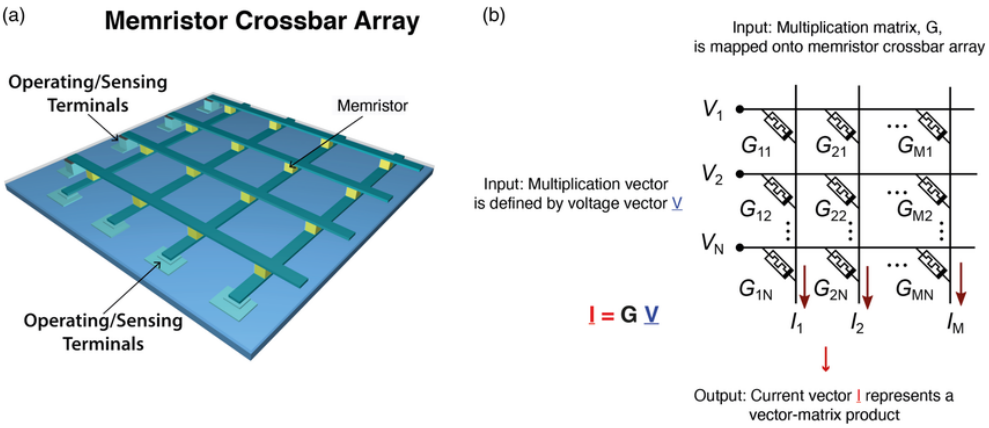


Figure 1.9 – a) Memristor crossbar array : memristors are placed at the intersection of input lines and output lines. b) Electrical schematic of a memristor crossbar array : inputs are presented as voltages, and the outputs are read as currents. Image from [46].

tage, resistance, and current. They can be scaled down to 2 nm for binary operations [47]. Memristor neural networks have a low energy consumption with only 1 pJ per writing operation [48]; this is the most energy-consuming operation of the network due to the neuromorphic architecture implementing in-memory computing.

Several challenges hinder the development of memristor neural networks. The main one is the fact that crossbar array geometry is prone to the appearance of sneak-paths [49]. The currents can travel through memristors, generating unwanted loss and perturbation that impact results. To counter this kind of issue, memristors are designed to have high resistances. However, with output current being smaller, the signal-to-noise ratio is degraded. Another challenge is to co-integrate efficient and compact neurons with these synaptic crossbar arrays[14]. Additionally, device variability remains a challenge to handle for large neural networks. Finally, the wiring of memristors and neurons is realized via metallic connections. This leads to complex connection patterns; moving to 3D geometry with crossbar array stacking [50] can offer some solutions but remains complex.

Impressive results have been recently obtained with memristor crossbar array based neural networks. A network of 1.4 million 90nm memristor synapses and 1600 neurons has been experimentally realized and was trained on-chip to classify 100 handwritten digit images from the MNIST dataset [34] with 92% accuracy [51]. Another example of powerful neural network achievement is the full hardware implementation of a 5-layer CNN on a chip composed of 8 arrays of 2048 memristors [52]. This network was able to solve the

MNIST task through ex-situ training followed by on-chip tuning, with more than 96% accuracy. Latest memristor networks use multi-core architecture and can reach large sizes, but are trained off-chip. An example is the 64-core chip fabricated by Wan et al. [53] that is able to process inputs at a speed of 63.1 TOPS and with an energy efficiency of 9.76 TOPS per W. Simultaneously, Ambrogio et al. realized a network with 45 million weights that was able to process inputs with an energy efficiency of 12.4 TOPS per W [54]. An architecture with 48 cores and 3 million parameters [55] achieved 99% accuracy on the MNIST handwritten digits classification task [34] and 84.7% accuracy on the CIFAR-10 [56] complex images classification task.

1.2.5 . Spintronics Neural Network

In the following sections of this thesis, spintronics for neuromorphic computing will be the focus and primary technology developed. Spintronics is the field describing the interplay between electronic and magnetic properties of materials and devices. It combines some of the best properties of optics and more standard CMOS or memristor devices. Magnetic properties can be manipulated with multiplexed radio-frequency (RF) signals, allowing a high degree of parallelism through frequency multiplexing, similar to optics. Additionally, spintronic devices are compatible with CMOS technology, using similar materials and requiring standard voltage and current levels to be controlled. Moreover, they can be fabricated at sizes down to 10nm [57], allowing for high density and low energy consumption. The energy consumption is also intrinsically reduced compared to standard resistive devices; indeed, manipulating the spin of electrons doesn't generate Joule's heating directly, allowing for less dissipation.

In this section, I will present various neuromorphic architectures developed for static inputs. The implementation of spintronic reservoir computing and other time-dependent neural networks will be presented in section 1.3.2. I will present several neuromorphic approaches based on spintronics and then focus more in-depth on radio-frequency-based spintronic architectures for neural networks developed in our team to introduce the key concepts for the next chapters.

Spintronic synapses. The non-volatility of magnetisation in spintronics can be harnessed to implement writable synapses of various types. The first building block that can be used in this way is the magnetic tunnel junction (MTJ) [58]. These devices are composed of a layer with pinned magnetisation and a layer with a free magnetisation spaced by a thin layer of insulator. Depending on the relative orientation of the free and fixed magnetisations, the electrons spin-polarised by the fixed layer will have more or less ability to transmit to the free magnetisation layer; the device will thus present high

or low resistance states. This constitutes the memory block of magnetoresistive random-access memory (MRAM). The resistance can be used as the stored weight of a synapse. A crossbar array implementation of 64x64 MTJ synapses was realised by Jung et al. [59] and was able to classify the MNIST dataset [34] with 93.23% accuracy. Another implementation with over 20,000 MTJs was realised [60] with above 95% accuracy on the same task.

Due to the maturity of MRAM technology, these synapses can be fabricated in large arrays; however, they have binary weights or, at best, multi-bit weights [61]. Analog synapses can be fabricated with spintronic components and effects. Magnetic domain walls can be used to realize synapses. The most straightforward implementation is to inject a domain wall into the free magnetisation layer. This domain wall can be moved by applying a writing current in order to move the wall. The resistance of the device can be tuned continuously by changing the fraction of magnetisation in the free layer parallel or anti-parallel to the fixed layer magnetisation. Such devices have been fabricated [62], [63][64], and networks implementing them have been simulated with success [65], [66].

Similarly, skyrmions can be used to realize memristive synapses. By accumulating skyrmions [67][68] or modifying the radius of a single skyrmion [69] [70] in magnetoresistive devices, the magnetisation—and thus the resistance—can be continuously tuned. Skyrmions are promising as they are nanometric in size and are topologically protected.

Finally, heterostructures of antiferromagnetic and ferromagnetic materials can also be used as synapses. When current is injected into the antiferromagnetic layer, the Hall's resistance of the heterostructure is modified [71]. This change of resistance is still present when the current is off, making this effect non-volatile.

Spintronic neurons. Some spintronic effects present non-linearities that can be harnessed to build neurons. One example is spintronic oscillator synchronisation. Depending on the frequencies of external inputs, coupled spintronic oscillators can present different synchronisation patterns. A network with 4 spin-torque nano oscillators (STNO) achieved classification of spoken vowels with high precision [72]. The input vowels were sent as input frequencies leading to different synchronisation states of the 4 oscillators. A similar approach was used with spin Hall nano oscillators for 2D pattern classification [73] [74]. Spintronics also facilitates the implementation of spiking neurons. These neurons integrate an input with a leak term and fire when the integrated value is above a given threshold. These so-called leaky-integrate-and-fire (LIF) neurons can be implemented with domain wall MTJ. The domain wall is shifted with input current pulses, and after enough pulses, the MTJ switches to a low resistance state, allowing the neuron to fire [75] [76]. Spiking neu-

ral networks use different concepts and training principles than the standard feed-forward or recurrent neural networks that I study in this thesis, nd I will not address them in the following.

Radiofrequency-based spintronic networks. Next, we will focus on radio frequency-based spintronic architectures for neural networks, which is the topic to which this work contributes. These networks are composed of radio-frequency synapses and neurons. To implement a neuron, a device needs to be able to receive multiple input signals, sum them, and output a non-linear transformation of this summed input. This can be achieved by using a spin-torque nano-oscillator (STNO) [77]. This type of component can emit an RF signal when a DC current is injected into it. This RF signal requires a threshold current to be generated and thus presents a non-linear relation versus DC current. Such behavior can be achieved using MTJs [78][79]. By injecting a low current into the fixed magnetisation layer, the electrons get spin-polarised, they tunnel through the insulator and exert a torque on the magnetisation of the free layer. If the current is strong enough, this torque will drive the magnetisation into precession, leading to the emission of RF power. MTJ-based neurons have already been demonstrated [80] and are one of the key building blocks of RF spintronic neural networks. A CNN with such spintronic neurons was proposed by Raimondo et al. [81] and achieved 98.92% accuracy on the MNIST dataset [34] and 91.89% accuracy on a more complex image dataset, FashionMNIST [82]. However, this study was realised purely in simulation and with mathematically defined synapses.

The second element to implement for a fully RF spintronics network is the synapse. Spintronic RF synapses are based on a resonant effect, the spin-diode response [83], whose physics will be developed in section 2. When an RF signal is injected near the resonance frequency of a ferromagnetic material, the magnetisation can be excited and driven into precession; this is the ferromagnetic resonance effect (FMR) [84]. When a magnetic material exhibits a magnetoresistive effect, it can, under certain geometric conditions, be excited by the FMR effect and lead to an AC electrical resistance oscillating at the frequency of the input RF signal. The mixing between the AC current and the AC component of the resistance in the material leads to a DC voltage generated when the input frequency is near the resonance frequency of the magnetic material [85]. The value of the voltage can be controlled by fine-tuning the difference between these two frequencies. This effect can be harnessed to create a synapse sensitive to an input at a given frequency that stores a synaptic weight in its resonance frequency. Leroux et al. demonstrated that chaining such synapses together permits the implementation of a MAC operation on frequency-multiplexed inputs [86], each addressed by a

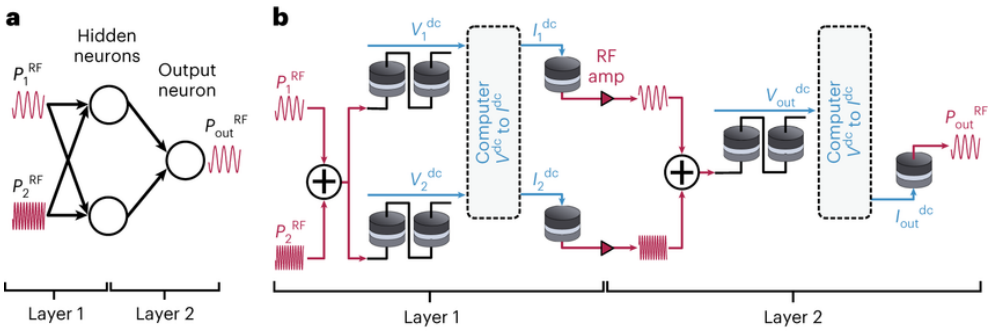


Figure 1.10 – a) Schematic of a 2-layer feed-forward neural network b) its spintronic implementation. Magnetic tunnel junctions are used as synapses when receiving RF inputs and neurons when receiving DC inputs. Image extracted from [80].

specific synapse.

With these two different building blocks, it is possible to build a spintronic neural network. The input vector has the form of frequency-multiplexed RF inputs, and weights are applied by synapses with matching frequencies. These synapses produce output voltages that can be converted to current and amplified before sending them to spintronic neurons. These neurons produce a non-linear response, outputting RF signals. By carefully choosing the properties of the neurons, they can output frequencies that can be injected into a next layer of synapses to create a multi-layer network. This kind of implementation has been realized in hardware [80] with a first layer of four synapses leading to two intermediate neurons; these neurons' outputs are then processed by two other synapses, leading to a final classification neuron. This architecture is displayed in figure 1.10. The power consumption of the building blocks of this network is estimated to be 10 fJ for a synaptic operation and 100 fJ for a neuron activation [87][80].

Additionally, an architecture designed for convolutional networks has been proposed by Leroux et al. [88] using similar components. This architecture is highly parallel and exploits the redundancy of the kernel weights to enhance the compactness of the network. It achieved 99.11% on the MNIST dataset in simulations. In my thesis, I demonstrate in chapters 2 to 4 that single-layer Permalloy devices can implement a convolution operation experimentally, following the concepts displayed in this article. This thesis presents the first experimental demonstration of the realization of a convolutional neural network with spintronic synapses processing RF signals, and its application to a complex and challenging task, here FashionMNIST [82].

1.3 . Networks using dynamics

In this thesis, I worked on using the dynamics of physical spintronic devices to implement a neural network. I will first present in this section the basics of networks capable of processing time-dependent inputs, such as RNNs and reservoir computing networks. I will then explain how the dynamics of a network can be trained either to process static outputs or to process time-series.

1.3.1 . RNN

Recurrent neural networks (RNNs) are a type of neural network designed to perform classification or prediction of time-dependent data. Contrary to the previously presented feed-forward, fully-connected, and convolutional networks, neurons in an RNN have recurrent connections ; in other terms, a neuron's output will be connected back to its input. As a consequence, the state of a neuron depends directly or indirectly on its previous value, introducing a time dimension. These types of connections were first introduced in 1982 in Hopfield's networks [89] ; these networks will be discussed in more detail in section 1.3.3. Following this work and the first explicit RNN designs [90][91], a general base structure of recurrent networks emerged. This architecture, consisting of hidden states with recurrent connections, is displayed in figure 1.11. Each neuron j of layer i has a hidden state that we will denote $h^n_{i,j}$. This state, for time step n, receiving an input $x^n_{i,j}$, evolves as follows :

$$h_{i,j}[n+1] = \sigma(W^{self}_{i,j,k} h_{i,k}[n] + W^{ext}_{i,j,k} x_{i,k}[n] + b_{i,j}) \tag{1.4}$$

There can be some variations of this equation ; the activation function σ can also be applied to $h_{i,j}$. In the case of a multilayer RNN, the input to a layer of neurons, $x^n_{i,j}$, is the outputs of the neurons of the previous layer $i-1$. This structure is displayed with a two layer RNN in figure 1.11, these two layers are linked by a matrix W^{ext}_1 so the input to the second layer is $W^{ext}_{1,j,k} h_{0,k}[n]$ or $W^{ext}_{1,j,k} \sigma(h_{0,k}[n])$ depending on the chosen implementation. The hidden states of the neurons accumulate the input values they receive ; this property permits them to keep a memory of previous inputs and thus to process non-trivial time-dependent data. RNNs are trained using backpropagation through time [92]. This technique, explained in detail in section 5.7, consists of unrolling the time dimension to backpropagate the errors through all previous time steps.

A slightly different framework has been proposed to implement neural networks capable of representing and modeling physical systems : Continuous Time RNNs (CTRNN) [93]. Time is no longer discretized in finite steps but is continuous ; the hidden state equation can now be expressed as a differential equation :

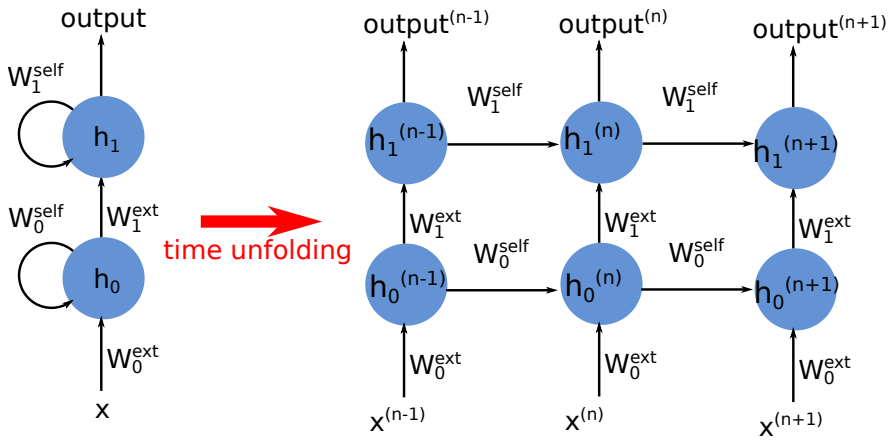


Figure 1.11 – Representation of a 2-layer recurrent neural network in its folded and unfolded form. Matrices W_{ext} implement connections between inputs and hidden states or between hidden states of different layers. Matrices W_{self} implement recurrent connections of hidden states.

$$\frac{dh_{i,j}(t)}{dt} = -\gamma h_{i,j}(t) + \sigma(W^{self}_{i,j,k} h_{i,k}(t) + W^{ext}_{i,j,k} x_{i,k}(t) + b_{i,j}) \tag{1.5}$$

The term γ is a damping term that helps prevent the divergence of the hidden state. We will discuss in more depth the link between CTRNN and the dynamics of physical devices in section 1.3.3.

More refined structures have been designed to improve memory properties, such as gated recurrent unit (GRU) networks [94] or long short-term memory (LSTM) networks [95]. These networks present more complex equations with parameters designed to remember long-term and short-term correlations of inputs. Current state-of-the-art models for processing natural languages, such as the chatbot chatGPT, are an evolution of LSTM networks. These networks are called transformers [96]; they can map an input sequence to a generated output sequence thanks to their encoder-decoder structure that extracts the meaningful correlations in sequences to generate a new one. They conserve some properties of LSTM, especially the attention, which defines the capacity to focus on some specific parts of a sequence.

1.3.2 . Reservoir computing

Reservoir computing was designed as a subtype of RNN [97]. The specific feature is that the network consists of a non-trainable part, the reservoir of neurons, and a trainable part, the output layer. As displayed in figure 1.12, the reservoir consists of numerous randomly connected neurons, including

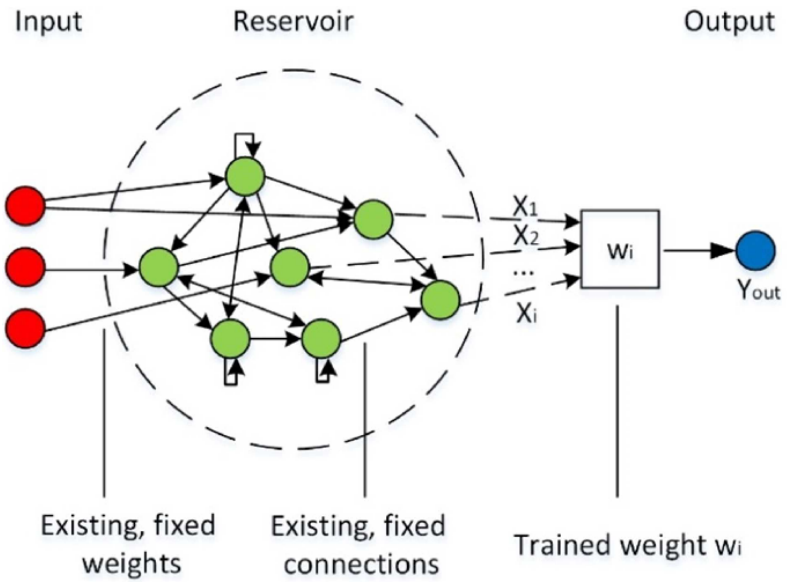


Figure 1.12 – Schematic of a reservoir computing network : inputs are sent to the reservoir via fixed connections, and inside the reservoir, random and recurrent connections lead to a dynamical response of the reservoir's neurons. A few nodes of the reservoir are chosen as output nodes, and trainable weights are applied to them to form a final output. These weights are optimized with linear algebra methods. Image from [100].

recurrent connections. The inputs are injected into some of the reservoir neurons ; the high and recurrent connectivity enables the reservoir to perform a complex non-linear transformation on the input. This is equivalent to projecting the input into a higher-dimensional space where it can be linearly classified. The final output layer consists of synapses connected to some neurons from the reservoir ; they are used to perform a final trainable linear transformation on this space to obtain the desired classification. The training is done by finding the best weights to match the outputs of the reservoir to the desired output ; this can be done with linear methods such as the Moore-Penrose pseudo-inverse [98] [99]. These one-step training methods require very low computational power.

Due to the recurrent connections in the reservoir, this type of network is suited to solve time-dependent tasks that require memory, where the inputs are sent sequentially into the reservoir. This approach is very interesting for hardware implementations as the reservoir can be directly realized with a physical system or media that present a non-linear response to an external stimulus and a complex time-dependent response.

1.3.3 . Neural network with trainable dynamics

As presented in the previous section, it is possible to exploit the dynamics of physical systems to solve cognitive tasks. However, due to the fact that only the last linear layer is trained in reservoir computing, this framework cannot be applied to solve complex state-of-the-art tasks. To try to overcome this limitation while conserving the memory and non-linear properties offered by reservoirs, a new concept has started to be developed : neural networks with trainable dynamics. These networks take inspiration from continuous time RNNs as they are expressed in terms of differential equations but generally aim to be implemented with physical devices with tunable dynamics. Dynamical networks can use the time dimension in two different ways : some networks use the time dimension as an additional depth, and others process time series with their dynamics. This difference is mainly due to the type of input to classify, either static ones or time series; in both cases, the networks have similar behavior.

The idea of using a time dimension to better classify static inputs was first proposed by Hopfield [89] in 1982. The proposed architecture consists of binary neurons all connected together, thus implementing recurrent connections. The goal of this network is to classify patterns correctly. When feeding an image, the neurons' states are sequentially evaluated until the full network reaches an equilibrium. The network is trained to reach different equilibrium states for different types of input patterns. In this framework, the neurons start from an initial state and evolve toward a final state suited for classification. This time evolution of input information is equivalent to the information propagation through a feed-forward network.

In 2019, Chen et al. proposed a general framework uniting the idea of evolving an input through time to classify it and a continuous-time approach. This model is called Neural Ordinary Differential Equation (NODE) [101]. In this architecture, a set of hidden states $\overrightarrow{h(t)}$ have a time evolution described by a differential equation controlled by a generic feed-forward neural network. This network takes $\overrightarrow{h(t)}$ as input and has parameters denoted θ_i.

$$\frac{d\overrightarrow{h(t)}}{dt} = f(\overrightarrow{h(t)}, \theta_i) \tag{1.6}$$

The initial state of the hidden states vector $\overrightarrow{h(0)}$ is the input value. This vector $\overrightarrow{h(t)}$ is evolved until an arbitrary time T, where the state $\overrightarrow{h(T)}$ is used for classification. This continuous evolution is represented in figure 1.13. The training of such a network can be done by backpropagation through time, unrolling the differential equation solver or by a method specific to differential equations, the adjoint method [102]. Neural ODE proved to be very powerful,

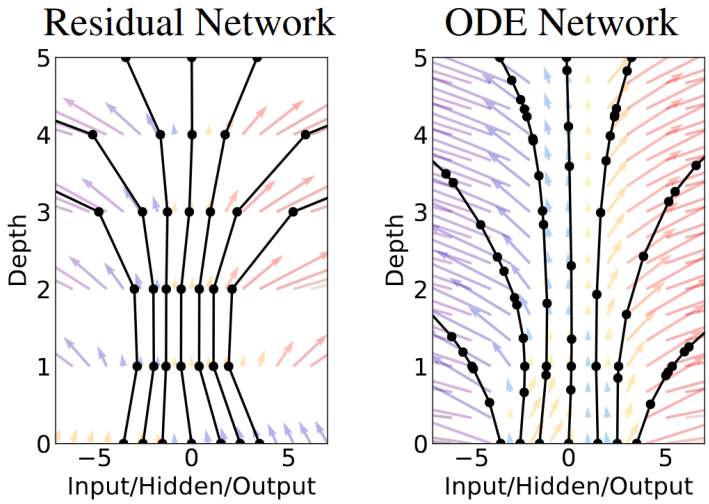


Figure 1.13 – Comparison between the flow of information in a state-of-the-art network (ResNet) and the Neural Ordinary Differential Equation framework. The layer depth of the residual network is a discrete equivalent of the time evolution of the NODE network. [101]

with performance comparable to state-of-the-art networks and reduced memory usage.

In the meantime, the idea of training coupled differential equations similar to NODE was also developed and adapted for time series classification. In 2020, Hasani et al. proposed a uniting framework suited to describe dynamic-based physical recurrent neural networks ; they introduced the concept of liquid time-constant (LTC) neural network [103]. These networks are a specific type of CTRNN that present naturally bounded dynamics, which is valid to describe physical systems. To implement this type of feature, the state of neuron i, $x_i(t)$, evolves according to the following equation :

$$\frac{dx_i(t)}{dt} = -\frac{1}{\tau}x_i(t) + f(\overrightarrow{x(t)}, \overrightarrow{I(t)}, \theta, t)(A_i - x_i(t)) \tag{1.7}$$

where $f(\overrightarrow{x(t)}, \overrightarrow{I(t)}, \theta, t)$ is a strictly positive function, $\overrightarrow{I(t)}$ is the input vector, θ is the vector of all weights of the neural network, $\overrightarrow{x(t)}$ is the state vector of all neurons and A_i is a positive term defining the upper boundary accessible to $x_i(t)$. This specific structure, inspired by biological neurons, ensures that all neuron states $x_i(t)$ in the vector $\overrightarrow{x(t)}$ have bounded values between 0 and A_i. Moreover, the relaxation time constant of each neuron is $\tau_{liquid} = \frac{\tau}{1-\tau f(\overrightarrow{x(t)}, \overrightarrow{I(t)}, \theta, t)}$, which depends on the input strength and trainable parameters θ. Thus, neurons can adapt their relaxation time to the input variation timescale to some extent through training.

These developments of neural networks controlled by differential equations led to the idea of using the differential equations naturally implemented by physical systems to realize hardware neural networks. These networks would compute directly through the time evolution of a physical system with minimal energy consumption.

1.3.4 . Hardware

A wide variety of hardware systems using the transient dynamics of physical devices have been proposed. I will not be exhaustive in this section; we will concentrate on implementations that leverage the transient dynamics of coupled physical devices. This field is under development and benefits from contributions from optics, acoustics, and spintronics; however, the paths and frameworks used are very diverse, which makes comparison and regrouping a complex task.

Before the implementation of physical networks with trainable dynamics, numerous hardware implementations of RNNs were realized. Examples can be found in optics [104], with FPGAs [105], or with resistive RAM blocks [106]. These architectures implement explicit recurrent loops with feedback loops that connect the output of a layer to its input, adding a time-delay. Meanwhile, physical reservoir computing has been developed to use the transient dynamics of diverse systems to compute. Implementations have been realized with different physical reservoirs; examples include optics with opto-electronic reservoirs built out of a single non-linear element with a feedback loop [107] or quantum systems with coupled quantum oscillators [108]. Here, we will do a short focus point on spintronic reservoir computing. Most reservoir computing approaches in spintronics are based on collective effects of complex magnetic textures. This type of complex media can be realized with skyrmions [109], spin ice [110], or spin waves [111]. Other approaches have been realized with individual magnetic devices, such as coupled spintronic nano-oscillators [112] [113] or ring nanowires with a moving domain wall [114].

While reservoir computing is efficient for simple tasks, more complex time series require more tunable networks such as CTRNN, where the dynamics can be trained. Several implementations of physical CTRNN processing acoustic waves have been tested in simulations. An example of an architecture was proposed by Hughes et al. [115]. This network is designed to classify recorded vowels; these inputs are sent as acoustic waves through a medium with locally tunable wave propagation speed. Classification is performed on the values of the wave at different points at the output of this trainable medium. The error is then backpropagated through the wave propagation differential equation

to optimize locally the propagation speed. Another implementation of a similar task was proposed, this time using acoustic resonators [116]. The acoustic signal of vowels is injected through a waveguide ; resonators are coupled to this waveguide and to one of two output waveguides. The signal is then recorded at the end of the input waveguide and of the output waveguides. Classification is performed on these output signals. The couplings of the resonators with the waveguides are the tunable parameters. Here, the time response of the resonators implements a dynamic response and memory similar to CTRNN. This network is also trained by backpropagation through time.

In spintronics and optics, simulations of dynamical networks that use the time dimension as a depth of processing have already been performed. An example of such a system was proposed by Rodrigues et al. [117]. In this article, a system of coupled STNOs is trained to classify the DIGIT dataset [118]. The STNOs' output RF powers are the internal states encoding information, and their initial states encode the input image. A set of control signals $A_{i,j}(t)$ controls in time the coupling between STNOs. They are the trainable parameters and can be optimized through a dedicated method : optimal control theory. This method is designed to control the dynamics of physical systems with time-dependent signals and has already been demonstrated as efficient for training the dynamics of an optoelectronic delay element [119].

This thesis will present developments that aim to process time series with a network of coupled STNOs rather than static inputs. This study is the first demonstration of a multilayer network of coupled oscillators trained through backpropagation through time. This training approach leads to better performance than reservoir computing and, coupled with the multilayer structure, fits in the framework of standard RNNs, allowing for future developments and scalability.

2 - A robust radio-frequency synapse based on the spin-diode effect in Permalloy

2.1 . Summary

The goal of this chapter is to demonstrate the realization of robust and easy-to-fabricate RF spintronic synapses. These synapses need to be capable of processing frequency-multiplexed RF inputs. We thus require devices that are selective in frequency and have a tunable response, ideally non-volatile. We chose to exploit the anisotropic magnetoresistive effect (AMR) to create such diodes. This effect converts RF power into a DC voltage whose amplitude depends on the input frequency and the device's resonance frequency. The difference in frequency is the weight stored by the synapse ; to change it, the resonance frequency of the device is tuned. This tuning is realized here by changing the local external magnetic field. We proposed and investigated three different synapse designs : one based on a single diode, one with a diode plus a current line to apply a local Oersted field, and the last design composed of two opposite diodes with a small shift in resonance frequency. This shift in resonance frequency is implemented by inserting the devices in a spatially-varying magnetic field, so that a small spatial shift between the two diodes translates into a shift of the applied external magnetic field, and therefore, of their frequencies. This last design presents advantages such as a sharp frequency profile and precise control of the synaptic weight. It is thus the chosen design for the following developments. This design, tested with NiFe5/Pt5 diodes of size 5x10 μm, implements a synapse capable of processing inputs in the GHz range with a non-volatile weight that is written via lithography. To fabricate these devices, we used UV lithography. A specific characterization setup was developed, integrating a source for RF inputs, a nanovoltmeter, and a set of magnets creating a controlled gradient of magnetic field that allows the magnetic field to be controlled by moving the devices in this field.

2.2 . Introduction

In this chapter, I will present how a thin film with anisotropic magnetoresistance can exhibit a spin-diode effect, and how this effect can be harnessed to apply a tunable synaptic weight as a multiplicative factor over an RF input. These diodes exhibit frequency selectivity on the input. I designed several synapse diodes with different geometries and advantages. These synapses will later be connected to implement a weighted sum.

2.3 . Physics of AMR-based Spin Diodes

2.3.1 . Anisotropic magneto-resistance

The anisotropic magnetoresistance corresponds to a dependence of the resistance of the material on the relative orientation of the current density and the magnetisation. This dependence can be expressed as : $R = R_0 + \Delta R cos(\theta)^2$[120], where θ is the angle between the magnetisation and the current density, R_0 is the resistance when $\theta = 90°$, and ΔR is the amplitude of resistance variation. This effect originates from an increase in the electron scattering rate when the current flow is parallel to the magnetisation. This is caused by a mixing, and thus a modification, of the "d" conduction orbitals by the spin-orbit coupling in magnetic transition metals [121].

2.3.2 . AMR Spin-Diode

The spin-diode effect is a voltage rectification effect that occurs when a magnetised material with a magnetoresistive effect is subjected to an RF signal at a frequency near its resonance frequency [85]. In our case, the magnetoresistive effect is the AMR effect, and the rectification effect occurs through the mixing of the RF current and RF-driven magnetoresistance. In the following, I will re-demonstrate the origin of the rectified voltage from the AMR effect, taking inspiration from the article [120].

Let's consider a thin film of Permalloy in the yz plane that represents our device. A fixed external magnetic field H_{ext} is applied in the z direction, as shown in figure 2.1, and is assumed to be strong enough to saturate the magnetisation along the z direction. The magnetisation M is defined as $\vec{M} = m_x\vec{e_x} + m_y\vec{e_y} + M_S\vec{e_z}$, where m_x and m_y are small variations of the magnetisation compared to the saturation magnetisation M_S. The AC current in the material generates an AC magnetic field h_{RF}. Here, I assume that the z component of this AC field can be neglected compared to the strong external field H_{ext} along this direction. When the RF current is injected near the resonance frequency, the magnetisation is driven into precession around the magnetic field at the injection frequency. The angle between the RF current and the magnetisation is thus a fixed component Φ_0 plus an oscillating component $\Phi_1(t)$, assuming that $m_x = 0$ as the driving field is in the sample plane.

Applying Ohm's law in our device, see equation 2.1, the voltage across the device is proportional to the oscillating angle $\Phi_1(t)$ multiplied by $I\cos(\omega t + \phi)$. The mixing of those AC terms leads to the rectification of a DC voltage.

$$V = R \times I \tag{2.1}$$

$$V = (R_0 + \Delta R cos^2(\Phi_0 + \Phi_1(t))) \times I cos(\omega t + \phi) \tag{2.2}$$

As the angle of precession Φ_1 is quite small it is possible to develop the cosine

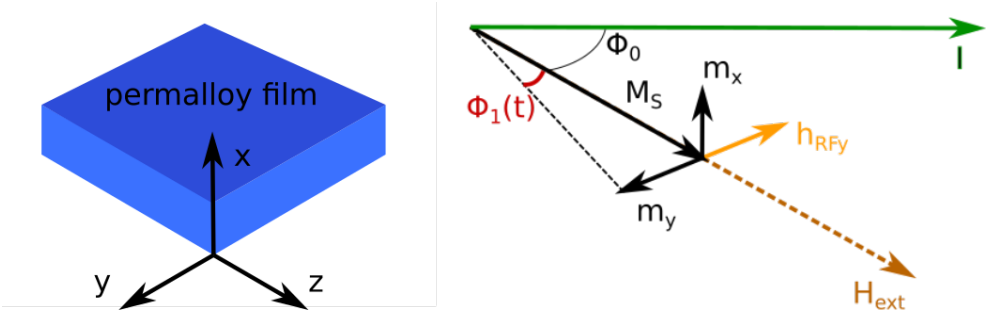


Figure 2.1 – Representation of the current flow and magnetisation is the sample, the angle between current and magnetisation as a fixed component Φ_0 and an oscillating one Φ_1(t). The main driving term of the magnetisation is h_{RFy}.

squared :

$$V = (R_0 + \Delta R(cos^2(\Phi_0) - \Phi_1(t)sin(2\Phi_0))) \times Icos(\omega t + \phi) \tag{2.3}$$

We can express the term Φ_1 as a function of the magnetisation : $\Phi_1 \approx \frac{m_y}{M_S}$. The term m_y can be expressed as a function of the external RF driving current. To establish this link, we start from the Landau-Lifshitz-Gilbert equation (2.4). This equation describes the dynamics of the magnetisation $\vec{M}$ subjected to a magnetic field $\vec{H_{total}}$. The term γ is the gyromagnetic ratio of the electron, and α is called the damping factor. The first term, $\gamma\vec{M} \times \vec{H_{total}}$, corresponds to the precession of the magnetisation around the magnetic field, and the second term, $-\frac{\alpha\gamma}{M_S}\vec{M} \times \frac{d\vec{M}}{dt}$, corresponds to the magnetisation relaxation towards the external field.

$$\frac{d\vec{M}}{dt} = -\gamma\vec{M} \times \vec{H_{total}} - \frac{\alpha\gamma}{M_S}\vec{M} \times \frac{d\vec{M}}{dt} \tag{2.4}$$

We consider our Permalloy film as infinite in the yz plane, as our diode designs have large dimensions, on the order of microns at the smallest. The x direction is considered finite as the sample thickness is on the order of nanometers. To compute the total magnetic field, we need to take into account the demagnetizing field, which has a contribution only along the x direction due to the small thickness of the sample and is expressed as $-(\vec{M} \cdot \vec{e_x})\vec{e_x}$. Two other contributions need to be taken into account : the external fixed magnetic field $H_{ext}\vec{e_z}$ and the RF field $h_{RFy}\vec{e_y}$, induced by the injection of AC current. The total resulting field is displayed in equation (2.5).

$$\vec{H_{total}} = H_{ext}\vec{e_z} - (\vec{M} \cdot \vec{e_x})\vec{e_x} + h_{RFx}\vec{e_x} + h_{RFy}\vec{e_y} \tag{2.5}$$

The LLG equation can be expressed in the form of a matrix equation :

$$\begin{pmatrix} \gamma(H_{ext}+M_S)+i\alpha\omega & i\alpha\omega \\ -i\alpha\omega & \gamma H_{ext}+i\alpha\omega \end{pmatrix} \begin{pmatrix} m_x \\ m_y \end{pmatrix} = \gamma M_S \begin{pmatrix} h_{RFx} \\ h_{RFy} \end{pmatrix} \tag{2.6}$$

This permits to link the magnetisation to the RF field through the susceptibility tensor χ, displayed in equation 2.8.

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \chi \begin{pmatrix} h_{RFx} \\ h_{RFy} \end{pmatrix} \tag{2.7}$$

$$\chi = \begin{pmatrix} \gamma H_{ext}+i\alpha\omega & -i\alpha\omega \\ i\alpha\omega & \gamma(H_{ext}+M_S)+i\alpha\omega \end{pmatrix} \frac{\gamma M_S}{\gamma^2(H_{ext}+M_S+i\alpha\omega)(H_{ext}+i\alpha\omega)-\alpha^2\omega^2} \tag{2.8}$$

In the AMR voltage, the term m_y can be replaced by $\chi_{yy}h_{RFy}$. Indeed, due to the reduced thickness of the film in the x direction, the term h_{RFx} can be neglected, and the value of m_x is negligible compared to m_y.

We can introduce angular frequencies for resonance condition ω_r, field ω_H and magnetisation ω_M :

$$\begin{aligned} \omega_r &= \gamma\sqrt{H_{ext}(H_{ext}+M_S)} \\ \omega_H &= \gamma H_{ext} \\ \omega_M &= \gamma M_S \end{aligned} \tag{2.9}$$

Using these new quantities and neglecting $\alpha\omega$ compared to $(\omega_H+\omega_M)$, χ_{yy} can be expressed in a simplified form visible in equation 2.10.

$$\begin{aligned} \chi_{yy} &= \frac{\omega_M(\omega_H+\omega_M+i\alpha\omega)(\omega_r^2-\omega^2-i\alpha\omega(2\omega_H+\omega_M))}{(\omega_r^2-\omega^2)^2+\alpha^2\omega^2(2\omega_H+\omega_M)^2} \\ \chi_{yy} &\approx \omega_M(\omega_H+\omega_M)\frac{(\omega_r-\omega)(\omega_r+\omega)-i\alpha\omega(2\omega_H+\omega_M)}{(\omega_r-\omega)^2(\omega_r+\omega)^2+\alpha^2\omega^2(2\omega_H+\omega_M)^2} \\ &\text{introducing the variable } \Delta H = \alpha\omega\frac{(2\omega_H+\omega_M)}{\omega_r+\omega} \\ \chi_{yy} &\approx A_{yy}\frac{(\omega_r-\omega)\Delta H-i\Delta H^2}{(\omega_r-\omega)^2+\Delta H^2} \end{aligned} \tag{2.10}$$

We obtain χ_{yy} as the product of a term $A_{yy} = \frac{\omega_M(\omega_H+\omega_M)}{\alpha\omega(2\omega_H+\omega_M)}$ which decreases with frequency of the R input and a second term $\frac{(\omega_r-\omega)\Delta H-i\Delta H^2}{(\omega_r-\omega)^2+\Delta H^2}$ which is a sum of a symmetric term and an antisymmetric one with respect to the resonance frequency ω_r. The symmetric term is a Lorentzian of the input frequency centered on the resonance frequency with width ΔH which is out of

phase with the magnetisation and the antisymmetric term is an antilorentzian of the same parameters and which is in phase with the magnetisation.

We can then replace the term m_y in the voltage by $\chi_{yy}h_{RFy}$, considering $h_{RFy} = h_{RF}\cos(\omega t)$. Then, taking the time average of the voltage, we observe that there is a remaining DC part; see equation 2.11. This is the spin-diode voltage. It is interesting to note that it conserves the symmetry of the term χ_{yy}, with the antisymmetric term being weighted by the cosine of the phase difference between the AC current and the magnetisation, and the symmetric one by the sine of this phase. Moreover, the voltage is proportional to $\sin(2\Phi_0)$, where Φ_0 is the mean angle between the external field and the magnetisation. Thus, in order to maximise the spin-diode voltage, this angle needs to be an odd multiple of $45°$. Finally, the AMR voltage is proportional to the input RF current squared, as the RF field is proportional to the RF current; thus, the voltage is proportional to the input RF power.

$$
\begin{aligned}
V &= (R_0 + \Delta R(cos(\Phi_0)^2 - \frac{\chi_{yy}h_{RFy}(t)}{M_S}sin(2\Phi_0))) \times Icos(\omega t + \phi) \\
V_{DC} &= -\Delta R\frac{A_{yy}}{2M_S}(\frac{(\omega_r - \omega)\Delta H}{(\omega_r - \omega)^2 + \Delta H^2}cos(\phi) + \frac{\Delta H^2}{(\omega_r - \omega)^2 + \Delta H^2}sin(\phi)) \\
&\times Isin(2\Phi_0)h_{RF}
\end{aligned}
\tag{2.11}
$$

With this theoretical analysis we see that we can use AMR to rectify an RF signal. In the next section we will explore how to use this effect to construct a synapse with tunable weight and frequency selectivity.

2.4 . AMR Spin-Diode Synapse Concept

A hardware synapse is a device capable of applying a stored weight value w on an input x, so that it outputs a signal proportional to the product $w \times x$. We can exploit the AMR spin-diode effect to build a spintronic synapse. Previous implementations have been realized with magnetic tunnel junctions [86]; however, the AMR spin-diode samples can be produced easily in our laboratory, allowing us to test complex designs with multiple robust synapses. As we can see in Figure 2.2 a), in the region around the frequency of resonance, the antisymmetric part of the AMR voltage can be approximated as linear with both the injected RF power and the difference between the resonance frequency of the device and the frequency of injection of the external input power. Indeed, when we consider $\omega_r - \omega$ to be small compared to the width of the spin-diode antilorentzian ΔH, the voltage simplifies to :

$$V_{DC} \propto \frac{\omega_r - \omega}{\Delta H} \sin(2\Phi_0) P_{RF} \tag{2.12}$$

The symmetric part of the AMR voltage is small in our samples and can be neglected; it only contributes as a small offset voltage. The resulting voltage can then be expressed as :

$$V_{DC} \propto (f_r - f) \times P_{RF} \tag{2.13}$$

Here, the RF power P_{RF} is the input value to the synapse, the detuning in frequency $(f_r - f)$ is the synaptic weight applied to the input, and the final DC voltage is the output value of the synapse. In this device, the RF input frequency is fixed; the synapse will address this input only if f is close enough to its resonance frequency. This frequency selectivity will enable frequency multiplexing as developed in the next chapter and demonstrated in previous works [83][86].

The weight is tuned by slightly changing the resonance frequency of the device to change the value of $(f_r - f)$, as displayed in Figure 2.2 b), c), and d), where the voltage is linear with power, and changing the magnetic field adjusts the slope of this linear dependence. To store this weight in the device, we need non-volatile writing of the resonance frequency of the device. It is important to note that the synaptic weight can be positive or negative with a single device, which is not the case for all hardware synapses; memristor synapses typically require two devices, one for positive weights and one for negative weights.

2.5 . Fabrication

2.5.1 . Spin diode sample design

The base spin diode sample design is displayed in figure 2.3. It is made of a stripe of magnetic material in our case Permalloy with gold connectors designed to inject current at a 45° angle with respect to the connector alignment in the Permalloy. This angle will be also the angle between the magnetic field and the current in the magnetic material. This permits to maximise the AMR voltage through the angular dependency.

2.5.2 . Lithography process

I fabricated all the samples with the same procedure, where patterning is done through optical UV lithography. The base film is a layer of 5 nm of nickel-iron capped with 5 nm of platinum deposited on a high-resistance silicon wafer. Diode shapes are patterned with a positive resist (SPR 700) by UV

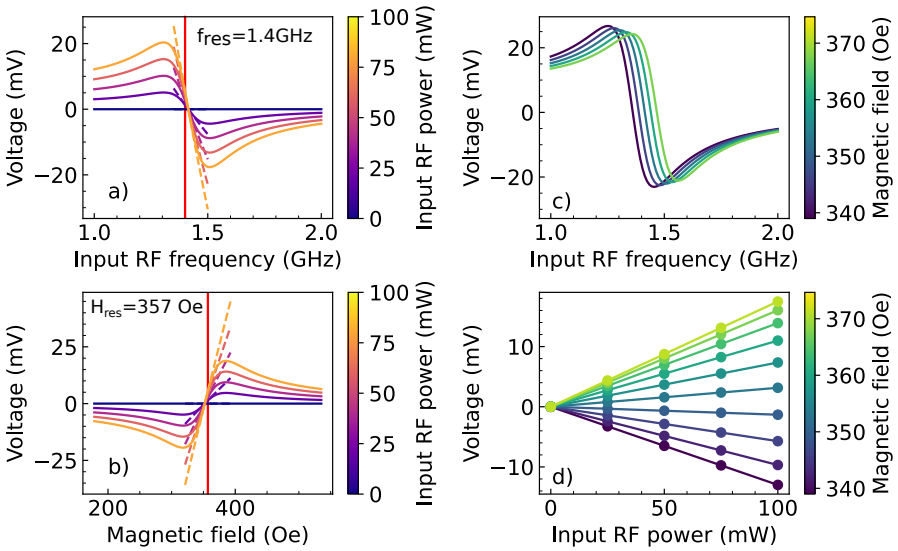


Figure 2.2 – a) Theoretical AMR spin-diode voltage versus input frequency under an external field of 357 Gauss while varying the input RF power from 0 to 100 mW. b) Theoretical AMR spin-diode voltage versus external magnetic field with an input frequency of 1.4 GHz while varying the input RF power from 0 to 100 mW. c) Theoretical AMR spin-diode voltage versus input frequency with a fixed input power of 100 mW while varying the external magnetic field between 340 and 380 Gauss. d) Theoretical AMR spin-diode voltage versus power while varying the external magnetic field at an input frequency of 1.4 GHz; this is the typical synaptic behavior. In the four plots, $\gamma = 0.01$, $\Delta H = \gamma \times 0.1$ GHz, $\omega_M = \omega_H = 2\pi \times 1$ GHz.

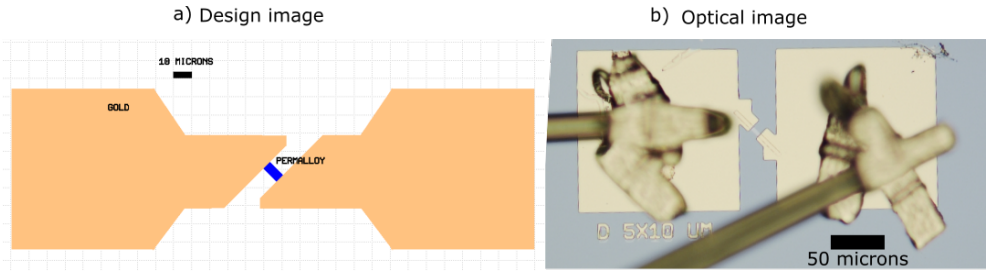


Figure 2.3 – a) Design of a Permalloy spin-diode with 5 microns width and 10 microns length b) optical image taken after fabrication

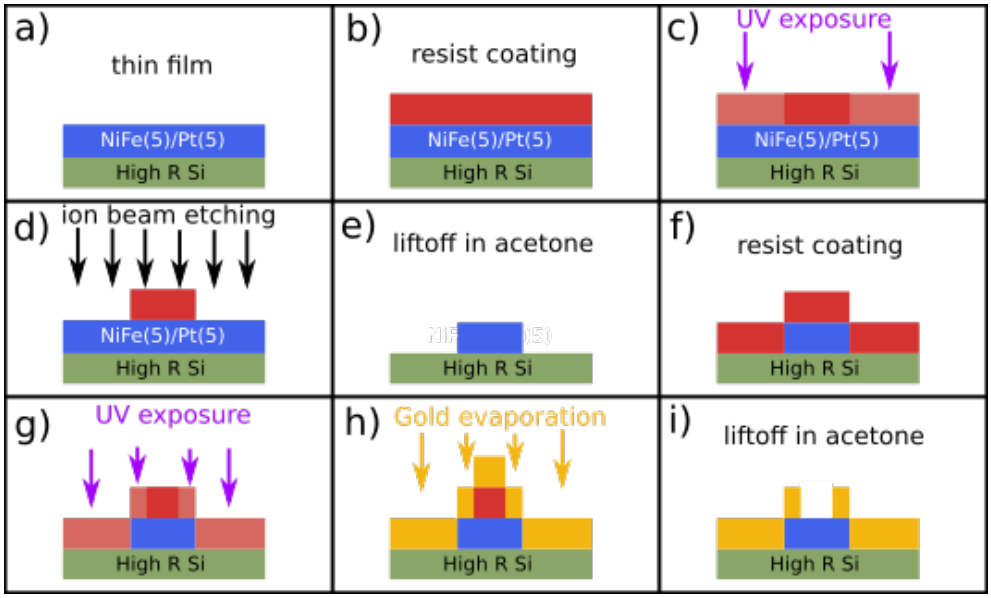


Figure 2.4 – Fabrication steps of the typical Permalloy spin-diode that I realised.

lithography. The resist is developed to protect only the metallic regions we want to keep. The sample is then etched with an ion beam etching machine to remove the magnetic NiFe layer outside the areas of the diodes that remain protected by the resist. These steps, displayed in figure 2.4 steps a) to e), pattern the shape of the diodes; we then need to pattern gold connectors for current injection. This time, the resist is exposed such that after development, all the sample is covered except for the region where the connectors are supposed to be. A gold evaporation is then done to deposit gold in these unprotected areas. The final step is to perform a liftoff of the remaining resist with acetone to keep gold only on the connector areas.

2.6 . Experimental Setup

To investigate the spin-diode effect, we need to design a specific experimental setup. The two main components of this setup are the RF source that supplies RF power to the spin-diode sample, and the nanovoltmeter to measure the response of the sample.

2.6.1 . Measurement Technique

In order to measure the spin-diode voltage, we need to disentangle this contribution from other contributions. First, we need to measure only the DC

voltage contribution and discard any AC contribution coming from the sample or from the RF source. This separation is made with a bias tee ; this component is mainly composed of two elements : a high inductance and a high capacitor mounted in a T geometry. The capacitor branch is made to block the DC current and transmit the AC signal ; the inductance branch is made to block the AC signal and transmit the DC signal. The RF power is injected through the AC branch into the AC+DC branch connected to the sample. Finally, the spin-diode voltage is measured through the DC-only branch. In order to discard any parasitic effect such as voltage offsets and drift, we measure the voltage with RF power on and without any power, with the difference of these two quantities being the spin-diode voltage. This is performed with a solid-state switch that can either direct the output of the source to the sample or to another channel. Additionally, an attenuator is placed before the sample. This attenuator reduces the cavity effect that occurs in the cable between the bias tee and the sample by attenuating multiple times the reflected amplitudes, thus reducing the noise at the resonance frequencies of the cavity observed in the spin-diode results.

2.6.2 . Field Gradient Implementation

Due to the uniformity and large size of our devices, our diodes do not have an anisotropy in the plane, which means that they all have similar base resonant frequencies. Therefore, in the following, the resonance frequency of the diodes is only controlled by the magnetic field that we apply to each of them, and not by the patterned geometry. In addition, since each diode represents a different synaptic weight, we will need to apply a different local field to each of them, in a very precise way. We have used permanent magnets to create the local field that will tune each diode to its base resonant frequency.

To have a controlled variable magnetic field created with permanent magnets, two options are possible : we can either have a magnet mounted on a motor and move it, or we can have fixed magnets and move the sample to different portions of the field. As we want to be able to have, at the same time, different magnetic field values in several devices on the same sample, we have implemented the second solution. By building a field gradient in one direction, two diodes can be placed on the same sample but spaced along this direction to experience different fields. This field gradient is realised with four identical magnets organized as shown in figure 2.5. At the center of the assembly, these magnets create a total field which is very small in x and y components and has a strong gradient along the z axis, as shown in figure 2.6. Consequently, our diodes need to be spaced along the z direction to be submitted to different, well-controlled fields. This field gradient has been calibrated using a 3D teslameter. The holder of the magnets, being in plastic,

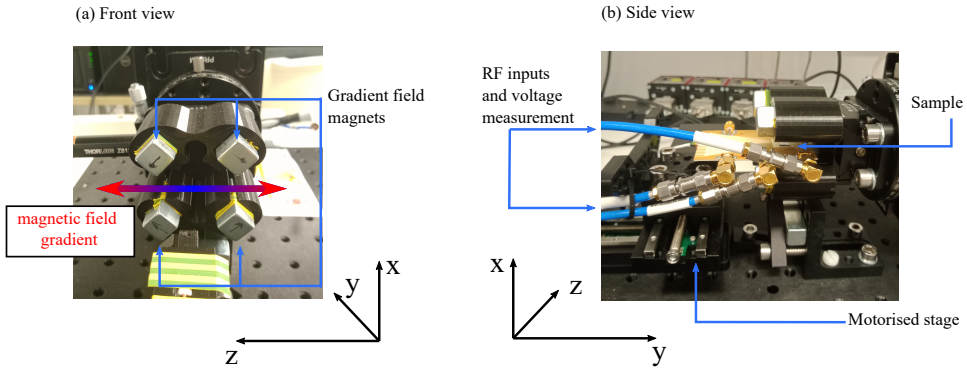


Figure 2.5 – a) Front view of the gradient magnets. The magnetic field is mainly oriented along the z-axis and has a strong gradient along the z-axis while being quite uniform in the y-direction.b) Side view of the sample placed between the four gradient magnets.

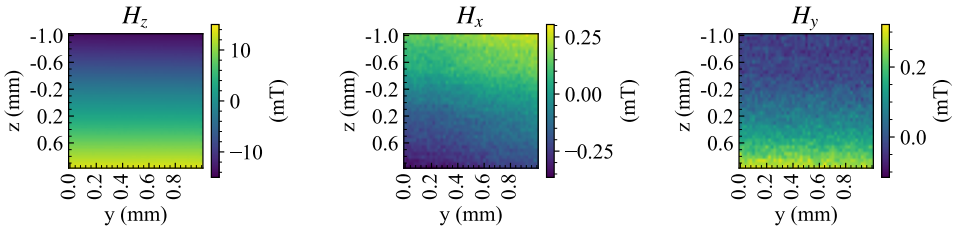


Figure 2.6 – Measurement of the magnetic field created by the 4 magnets along x, y, z with a 3D teslameter

experiences some deformation caused by the repulsion of the magnets; this deformation causes the appearance of a small H_y component nearly constant in the space between the magnets. This contribution is cancelled with a fixed magnet placed below the four magnets visible in figure 2.5. In this configuration, we obtain a gradient of the magnetic field component H_z along z of 140 Oe/mm; this strong gradient allows us to obtain different fields in closely spaced devices.

2.7 . Experimental results

2.7.1 . Experimental spin-diode measurement, synaptic behavior

In this section, the measured spin-diode response of a device with 5 microns width and 10 microns length is displayed. The device is placed between the four gradient field magnets such that the field is in the zy plane of the

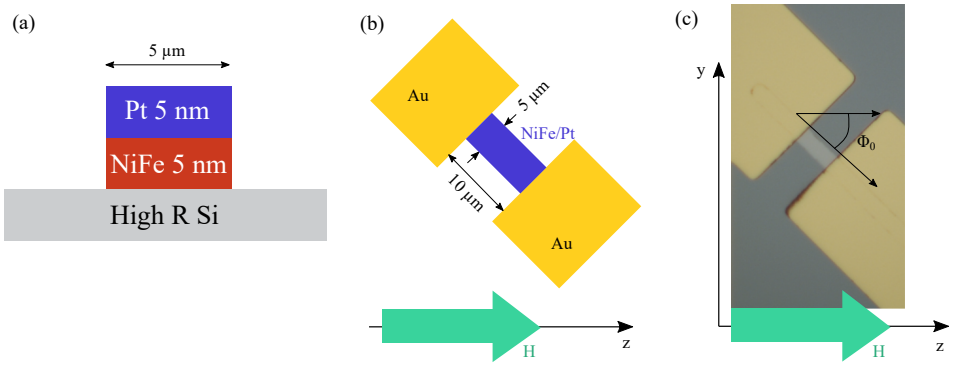


Figure 2.7 – Spin diode stack (a) and geometry (b)(c).

magnetic thin film and aligned along z. The current flows in-plane with a $\pm 45°$ angle to the magnetic field, maximizing the AMR voltage. This geometry is displayed in figure 2.7.

The DC voltage is recorded as a function of the frequency for different magnetic field intensities. These different magnetic field intensities are achieved by changing the position of the sample along the z-axis. These experimental results are displayed in figure 2.8 a). We observe the rectified voltage V_{DC} of the spin-diode effect, with its recognizable antilorentzian contribution. The resonance frequency can be estimated as the center of the antilorentzian where the derivative of voltage versus frequency is maximum in absolute value. This resonance frequency shifts with the field and can be plotted for both positive and negative fields; see figure 2.8 b). This dependence can be fitted using Kittel's law, equation 2.9, from which we can extract an effective magnetisation $mu_0 M_s$ of around 9.68 kOe. We also observe, as expected from the spin-diode voltage in equation 2.11, that the amplitude of the rectified voltage decreases when the resonance frequency of the magnetic material increases. This decrease is not exactly at the theoretical rate; as we will see in more details in chapter 3 section 3.5.2, this is due to the impedance mismatch created by the wire bonding connecting the sample and the RF waveguides on the sample holder.

In figure 2.9 a), the spin diode response as a function of the injection frequency is displayed for a fixed field and varying the input power; in plot b), the response versus field is displayed for a fixed frequency and varying the input power. In both cases, we see that the amplitude of the rectified voltage increases with the input power. In figure c), the DC voltage is displayed as a function of the input RF power for a fixed input frequency of 2.2 GHz and for a fixed external field close to the resonance field. The DC voltage varies linearly with the input RF power, and the slope of this dependence can be tu-

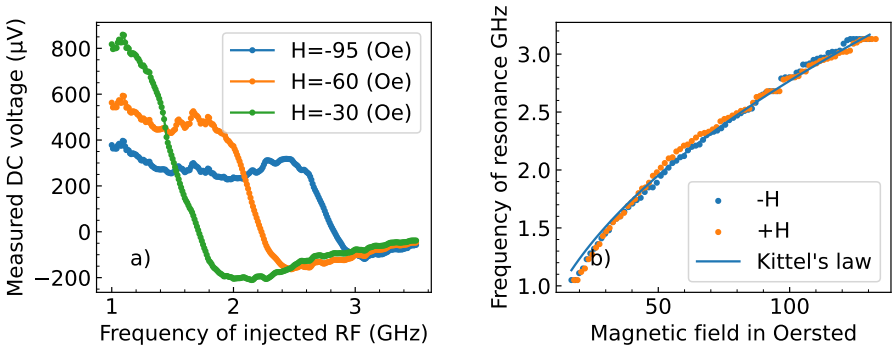


Figure 2.8 – a) Measurement of the frequency profile of the spin-diode voltage for different magnetic field intensities. b) Extracted frequency of resonance versus magnetic field for the Permalloy spin-diode with 5x10 microns geometry. The fitted Kittel's law gives us an effective magnetisation $mu_0 M_s$ of 9.68kOe.

ned by changing the external magnetic field via the z position between ± 15 Oe around the resonance field . This is the characteristic and expected synaptic behavior $V_{DC} \propto (f_r - f) \times P_{RF}$. Some slight non-linearity arise for fields close to the resonance field. This can be an effect of the lorentzian contribution that is more visible when the antilorentzian term is small, or a shift of the resonance frequency as the input RF power increases. This second hypothesis is more likely, as this effect was observed preferentially in small devices where the density of AC current is higher, increasing heating or other non-linearities with power.

As the synaptic weight is fixed by the device's resonance frequency, the tuning of this synaptic weight is achieved by changing the position of the diode along the z axis, hence modifying the strength of the magnetic field applied to the device.

2.7.2 . Synapse with current lines

In the previous design, the synaptic weight is tuned by changing the diode position in an external magnetic field. This method is compatible with the off-chip determination of the weights before nanofabrication that we will employ for convolutions. However, it doesn't allow for a local control of the weights of several diodes on the same chip for fine-tuning or on-chip training, as their positions cannot be tuned individually after fabrication. In this section, we evaluate a diode design allowing on-chip tuning of weights through the application of local Oersted field.

In order to enable individual updates of weights after fabrication, an ad-

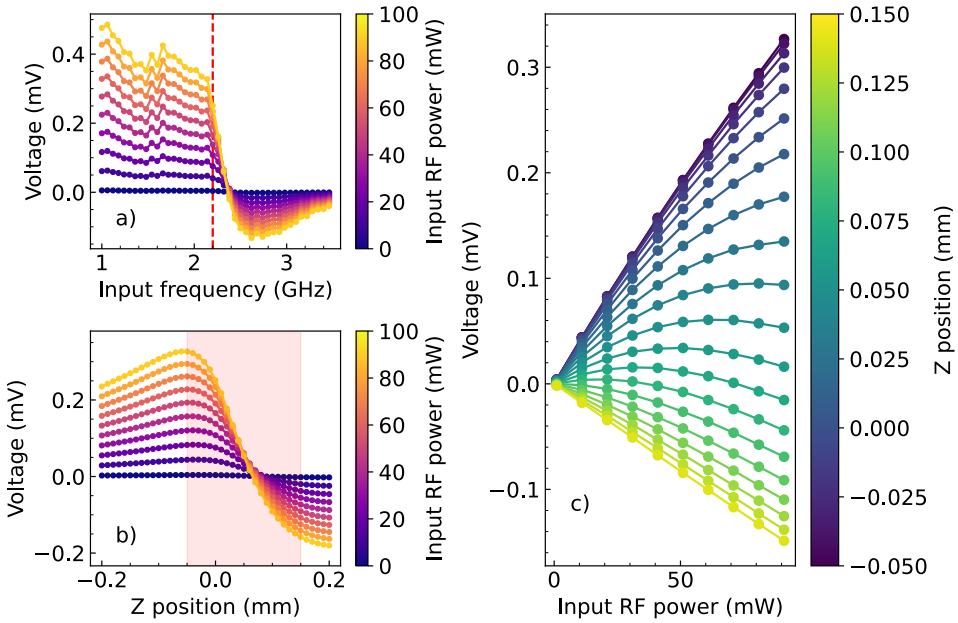


Figure 2.9 – a) Experimental AMR spin-diode voltage versus input frequency while varying the input RF power from 0 to 100mW. b) Experimental AMR spin-diode voltage versus the z position of the sample while varying the input RF power from 0 to 100mW. c) Experimental AMR spin-diode voltage versus power while varying the z position of the sample in a range of 0.2mm with an input frequency of 2.2 GHz, this is the typical synaptic behavior.

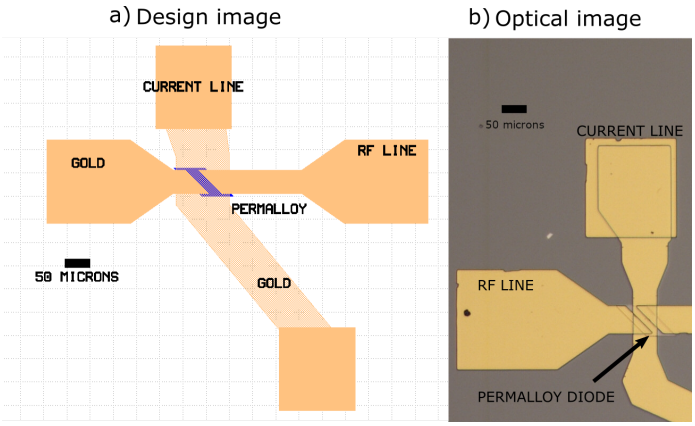


Figure 2.10 – a) Design of a Permalloy spin-diode with 70 microns width and 20 microns length. A current line is patterned on top to create an additional Oersted field in the magnetic layer to implement the synaptic weight. b) Optical image taken after fabrication.

ditional current line was patterned above the spin-diode device with an oxide spacer between the two; this design is displayed in figure 2.11. The oxide spacer is made by atomic layer deposition to obtain an HfO_2 layer of 45 nm; the gold current line is then patterned on top with another UV lithography step similar to the gold connector patterning described earlier.

When a DC current is sent in the current line, an additional Oersted field is created in the device, allowing for individual weight tuning. By shifting the resonance frequency of the diode, this allows us to tune the voltage linearly; the output voltage can be re-expressed as $V_{DC} \propto I_{DC} \times P_{RF}$, where the synaptic weight is I_{DC}. The experimental demonstration of this weight tuning is displayed in figure 2.11. In sub-figure a), the AMR voltage frequency profile is displayed while varying the input power; the voltage appears proportional to the power. Sub-figure b) displays the voltage at a fixed input frequency of 2.56 GHz while varying the DC current in the stripline and the RF power. The voltage is quite linear with the DC current intensity. Finally, the synaptic behavior is displayed in sub-figure c); the output voltage varies linearly with the input RF power, and the slope of this dependence—corresponding to the synaptic weight—can be controlled also linearly with the DC current applied in the stripline.

I managed to obtain broad tuning of a single synaptic weight; however, several drawbacks arise due to the current lines. First, the current lines need to be constantly powered to create the additional Oersted field corresponding to the desired weight; this leads to both volatility of the weights and energy dissipation. Additionally, the current lines are shunting a part of the RF power

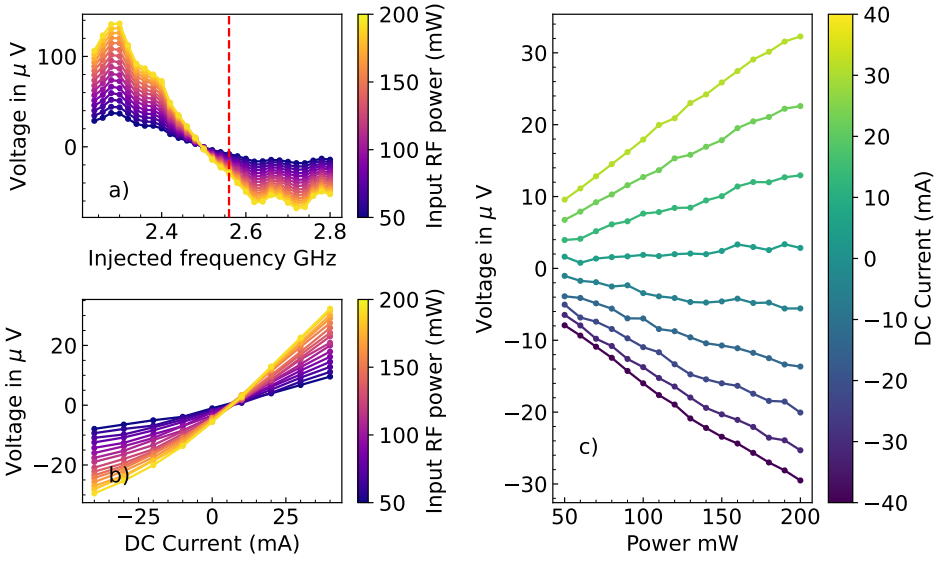


Figure 2.11 – a) Experimental AMR spin-diode voltage versus input frequency while varying the input RF power from 0 to 100 mW. b) Experimental AMR spin-diode voltage versus intensity in the current line while varying the input RF power from 0 to 100 mW. c) Experimental AMR spin-diode voltage versus power while varying the intensity in the current line in a range of ± 40 mA with an input frequency of 2.56 GHz; this is the typical synaptic behavior. It is important to note that this diode has a different geometry of 70x20 microns, leading to a much smaller density of current and thus a smaller rectified voltage.

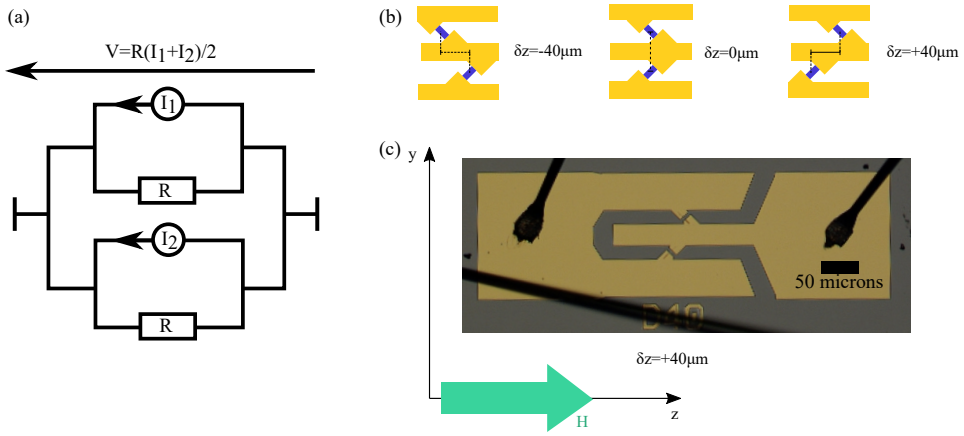


Figure 2.12 – a) Electrical scheme of a synapse made of two Permalloy spin-diodes with 5 microns width and 10 microns length connected in parallel ; each synapse can be considered as a current source. b) Design of synapses with different δz. c) Optical image taken after fabrication.

sent to the diode through capacitive coupling across the oxide spacer ; this is a big issue for scaling up. Indeed, when connecting several diodes in series, each current line will pick up a part of the RF power flowing from diode to diode, the final one receiving only a fraction of the input RF power and thus its signal being very small compared to the first ones. Finally, the fabrication proved quite challenging, as the current lines tended to have sticking issues on the oxide, and the oxide was sometimes too thin on edges, leading to short-circuits between the diode and its current line. All these drawbacks convinced us to move to a third architecture based on the first one, where the weights are implemented during the lithography, as detailed in the next section.

2.7.3 . Double diode synapse

The single device synapse design presents an important drawback : the frequency profile of the DC voltage presents an important background outside of the resonance frequency region. This background is not an issue when dealing with a single synapse. However, when we want to have several synapses at different, closely spaced resonance frequencies, the overlap of several backgrounds will add some unwanted contributions. These additional voltages are hard to take into account precisely. In order to discard these backgrounds, I designed a new architecture of a synapse made of two diodes in parallel with opposite signs ; see figure 2.12. When the two diodes are submitted to the same magnetic field, the AMR spin-diode currents cancel each other. By introducing a difference of magnetic field between the two devices, the total signal becomes non-zero around the resonance. This signal is equivalent to a differential measurement of the single device, creating a peak cen-

tered at the resonance frequency and reduced background outside the resonance ; see figure 2.13, where the signal of two diodes is predicted based on an interpolation of a single diode behaviour as a function of its z position. The peak amplitude can be tuned by changing the field difference between the two diodes, allowing for both positive and negative amplitudes. In this case, the AMR voltage can be expressed as $V_{DC} \propto \delta H \times P_{RF}$, where δH is the field difference between the two diodes. In my setup, a field difference can be implemented just by introducing a shift in the z position of the diodes due to the gradient of the magnetic field in this direction, thus $V_{DC} \propto \delta z \times P_{RF}$. This also presents another advantage : the δz term corresponding to the synaptic weight is a local quantity stored at the device level as a relative position, contrary to the global field value in the single device design.

I demonstrated the concept of a double diode synapse by measuring the response of synapses with the design represented in figure 2.12, with a δz of $+40\mu m$, $0\mu m$, and $-40\mu m$. The resulting curves are displayed in figure 2.14. In sub-figures a), b), and c), the frequency responses are displayed while varying the input RF power. We observe the desired properties : a single narrow peak with no additional background. The amplitude of the peak is controlled by the shift δz. Indeed, when $\delta z = \pm 40\mu m$, we get a positive or negative peak of the same amplitude of 60 μV, and when $\delta z = 0$, the peak has a negligible amplitude. In sub-figure d), the synaptic behavior is displayed : the output voltage is linear with power, and the slope of the dependence, i.e., the synaptic weight value, is linear with the shift δz.

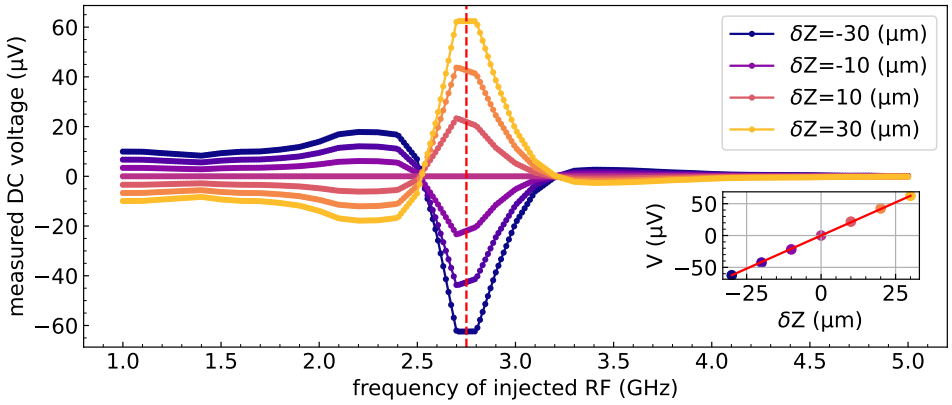


Figure 2.13 – Frequency response of a synapse made of two spin-diodes with a position shift and opposite signs, based on an interpolation of the response of a single device. The position shift is equivalent to a field shift; with zero shift, the AMR signals cancel each other, and with a non-zero shift, we can obtain a positive or negative narrow peak that is tunable in amplitude. The amplitude of the peak is linear with the shift.

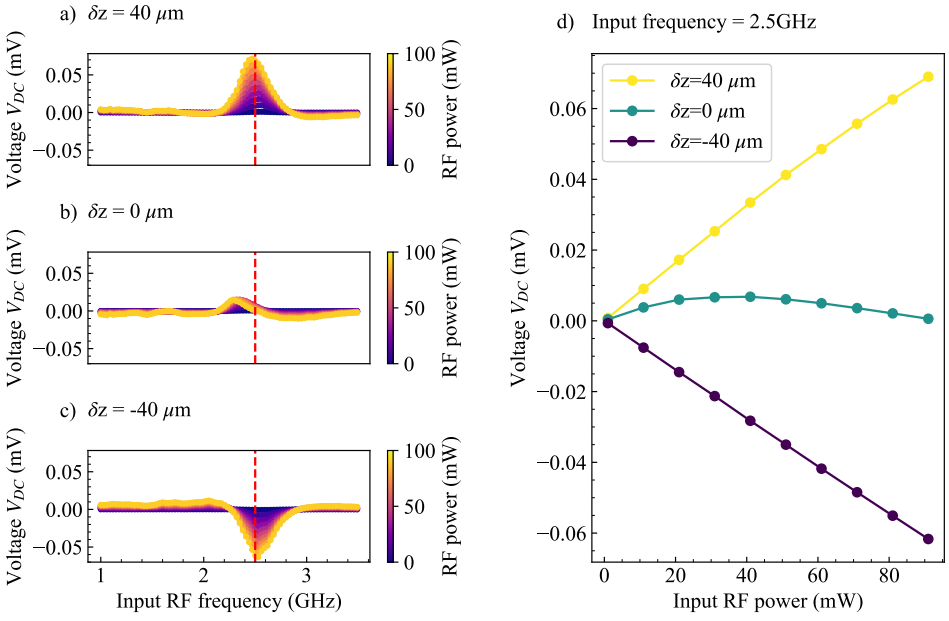


Figure 2.14 – a) Frequency response of a double diode synapse with a shift of +40 microns while varying the power from 0 to 100 mW. b) Frequency response of a double diode synapse with a shift of 0 microns while varying the power from 0 to 100 mW. c) Frequency response of a double diode synapse with a shift of -40 microns while varying the power from 0 to 100 mW. d) Synaptic behavior for these three different synaptic weight values.

2.8 . Conclusion

In this section, I demonstrated that the AMR spin-diode voltage can be considered as a synaptic output, where the input is an RF input power and the weight is the small frequency detuning between the input and the resonance frequency of the spin-diode device. This synapse has the advantage of being frequency selective. I built a spin-diode measurement setup with a unidirectional in-plane field varying along the z axis. I designed and tested three different types of synapses : one where the resonance frequency is tuned by moving the device along the z axis to change the magnetic field ; one where the magnetic field is tuned with a current line patterned on top of the spin-diode device ; and finally, one where two spin-diode devices are in parallel with opposite signs and a small shift in position, thus in resonance frequency, where the sign and amplitude of the position shift implement the weight. This last design is the one chosen for combining several diodes, as this differential geometry cancels unwanted background contributions outside of the resonance region while implementing a non-volatile weight in the lithography. In this configuration, weights cannot be tuned after fabrication ; however, the well-defined frequency response allows for precise control of the synaptic output.

3 - A chain of spin-diodes

3.1 . Summary

The goal of this chapter is to fabricate chains of synapses suited for precise multiply and accumulate operations. These chains need to feature synapses that process inputs at different frequencies. The synapses in these chains need to have a linear response when subjected to multiple inputs. Moreover, the weights implemented at each input frequency need to be precisely controlled, taking into account the contribution of the addressed rectifying synapse but also small contributions from synapses at neighboring frequencies. In this chapter, I present the experimental verification of the linearity of the previously fabricated synapses. To implement different base frequencies in each synaptic diode, the synapses are organized into chains placed into the previously demonstrated field gradient. Each diode is at a different position along the field gradient axis, thus it is subjected to a more or less intense field and has its own resonance frequency. Chains with a mixed parallel-series configuration for synapses were designed, allowing for impedance matching at 50 Ohms. To obtain the desired weight in a chain, we designed and measured a reference diode that is used as a model for all diodes in a chain. From this individual diode model, given that all diodes are nominally identical, we can model a full chain. This model is used to find the best spatial configuration of diodes to obtain synapses with the correct resonance frequencies and synaptic weights using numerical optimization methods. With this design, we fabricated 3 chains with 4 synapses each. Synapses are spaced at distances around 100 to 200 micrometers, corresponding to frequency shifts from 0 to 5 GHz, and have internal spatial shifts around $\pm 50\ \mu m$ to implement weights. The resulting weights after fabrication present a mean error of 5.2%

3.2 . Introduction

In the previous chapter, I demonstrated how a simple permalloy spin-diode can be used as a synapse processing RF inputs and producing a DC voltage output. In this chapter, I will demonstrate how we can chain these devices in a controlled way to produce a weighted sum operation on frequency-multiplexed inputs. First, I will explain the chain design and its constraints in frequency and geometry, then I will present how to design the correct geometry for a given frequency response. Finally, I will compare this response to the experimental one.

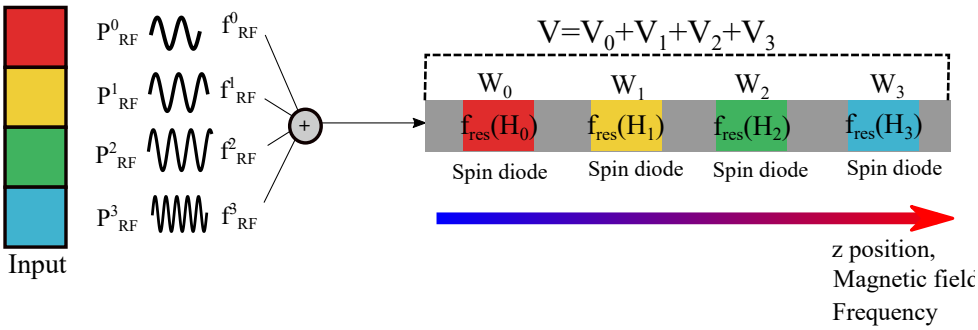


Figure 3.1 – Representation of the spintronic MAC. Each of the four synapses with different resonance frequencies in the chain will address only the input with a matching frequency, applying its stored weight on the RF power. The resulting voltages from each synapse are naturally summed when measuring the voltage across the chain.

3.3 . Spintronic Multiply and Accumulate Operation

The input sent to a neuron in a network is the weighted sum of the outputs of neurons in the previous layer; the weighting is done via several synapses. In the case of spintronic devices, weighting is performed by the frequency response of each diode, and summing is achieved by connecting diodes in series [83][86]. The weighted sum, also called multiply and accumulate operation, with a chain of four spintronic devices is displayed in figure 3.1. The goal is to apply weights w_0, w_1, w_2, w_3 on inputs P_0, P_1, P_2, P_3 and sum these four terms $w_i P_i$. Each input is encoded as an RF power at a different frequency f_i; these inputs are then combined and sent to the chain. Our chain is made of four diodes that will each apply the desired weight on one of the four inputs. The selection of the input processed by a synapse is done by matching its resonance frequency to the frequency of the input. If the resonance frequency of a synapse matches the frequency of an input, the synapse will produce a DC voltage $V_{DC} = P_i w_i$. If the resonance frequency isn't close to the frequency of the input, then the rectified voltage becomes negligible, especially in the case of double diode synapses. Finally, the total voltage across a chain is, due to Kirchhoff's voltage law, the sum of all voltages produced by each diode; thus, $V_{chain} = \sum_i P_i w_i$.

This multiply and accumulate operation has been demonstrated experimentally with two magnetic tunnel junctions [86]. In this section, we will propose designs to implement this operation in the most efficient and scalable way with Permalloy synapses.

3.4 . General Chain Design

Chains capable of performing MAC operations require several properties. They need to exhibit a linear response when subjected to multiple inputs in order to perform summation. Each synapse in the chain needs to have its own controlled resonance frequency, and ideally, these frequencies should be as close as possible to maximize the density of devices in the accessible frequency range. Finally, the total resistance load of the chains must be kept at a reasonable value, ideally 50 Ohms, no matter what the number of devices is. We will develop these different points in the following subsections.

3.4.1 . Verification of Linearity

A first necessary test is to check that diodes don't exhibit non-linear effects that would lead to a shift of their resonance frequency at high input powers, thus modifying the frequency bandwidth they are supposed to process in an undesired way. To test this property, I injected two frequencies, f_1 and f_2, either separately or simultaneously while recording the voltage. When sending the two tones simultaneously, the total measured voltage should be equal to the numerical sum of the voltages recorded when sending both frequencies individually in the absence of non-linear effects. This is indeed what we observe in figure 3.2; the voltage map from sending both signals at the same time a) is identical to the map created by numerically adding the two frequencies' voltages b). The difference between the two signals displayed in c) doesn't present any pattern, even when the two frequencies are close. The differences are only due to noise, as visible in plot d).

3.4.2 . From spatial distribution to frequency distribution

As described in section 3.3, in order to perform a MAC operation on frequency multiplexed inputs, we need to have a chain of devices with different resonance frequencies. Here, we combine the field dependence of the resonance frequency of our Permalloy diodes with the field gradient described in section 2.6.2 to create chains of devices with varying resonance frequencies. As illustrated in figure 3.3, we can define the resonance frequency of each diode by placing it at a particular spatial location along the z axis, the axis that has a strong gradient in field.

This approach allows us to conveniently map spatial location to resonance frequency, enabling us to directly define the resonance frequency of each device with very high resolution during lithography. Given a spatial resolution of 2 μm for our UV lithography system, a field gradient of 13.87 $\mu T/\mu m$, and a typical frequency dependence of 175 MHz/mT, using this approach, we can define the resonance frequency of each diode with a typical resolution of 4.9 MHz.

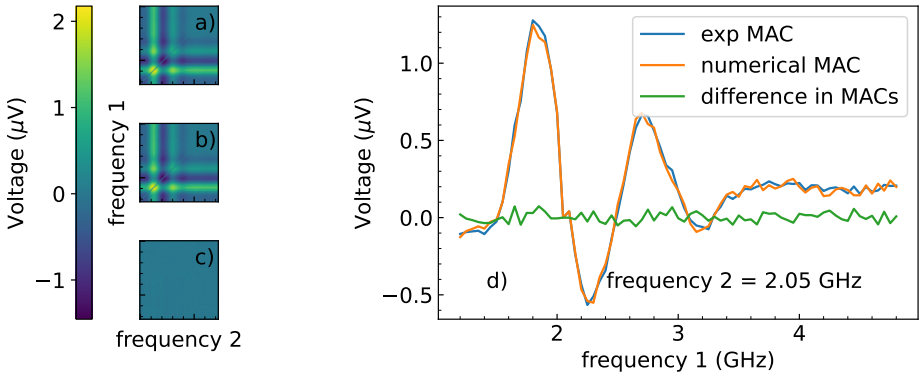


Figure 3.2 – a) Spin-diode voltage of a chain when sending two input frequencies simultaneously. b) Numerically summed spin-diode voltages of a chain when sending two input frequencies separately. c) Difference in spin-diode voltage between the numerically summed voltages and those summed via chaining geometry. d) Spin-diode voltages when numerically summed and summed via chaining geometry at a fixed second input frequency. The absence of difference between the two voltages proves the linearity of our devices, which show no noticeable non-linear frequency mixing effect.

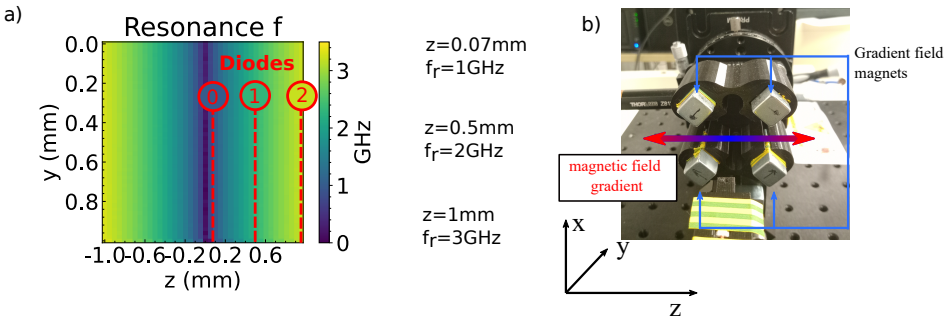


Figure 3.3 – Each synapse of a chain needs to be placed at the right position in the magnetic field to have the correct resonance frequency. a) Synapses are spread along the z axis of magnetic field gradient ranging from low to high frequencies. b) Array of magnets creating the magnetic field gradient along z.

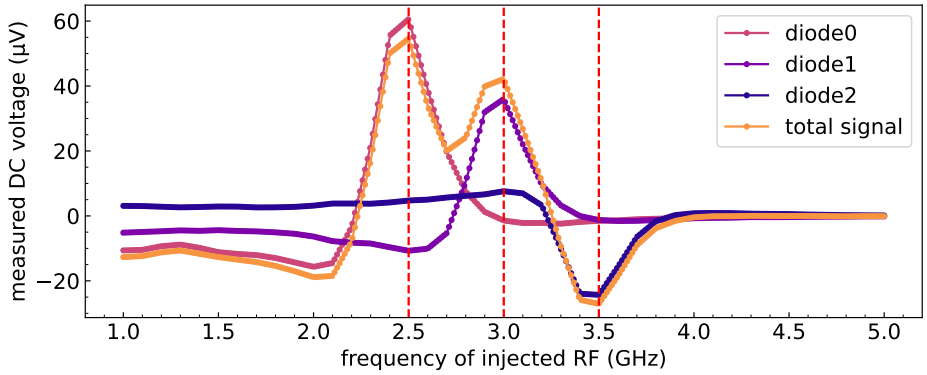


Figure 3.4 – Interpolated individual synapses signals in frequency and total chain signal for 3 synapses. The resonance frequencies of synapses can be as close as 0.5 GHz adding only small predictable overlap contributions.

3.4.3 . Minimum frequency spacing

When building a chain, we need to ensure that we exploit the available frequency range optimally. Since the synapses don't exhibit non-linear effects, the main constraint in frequency spacing is the overlap in frequency of the synaptic peaks. In figure 3.4, the individual responses of three synapses and the total response for a chain of these three synapses are represented. We can see that the total chain signal at 2.5 GHz is mainly caused by diode 0, whose frequency of resonance is 2.5 GHz. However, neighboring diodes 1 and 2 also contribute by small amounts to the total chain signal at this frequency. Ideally, this overlap could be taken into account during the final design phase and would not be an issue; however, this requires a very precise knowledge of the frequency response of a synapse, which can be a challenge due to parasitic contributions to the frequency response of a chain caused, for example, by the inductance of wire bonds. We minimize neighboring synapses' contributions by using the double diode synapse design, which presents a peaked response in frequency with low or no background. Additionally, we maximize the diode spacing within the available bandwidth. In our case, the custom multi-channel RF source that will be presented in the next chapter covers the bandwidth between 1.2 GHz and 3.3 GHz. As can be observed in figure 3.4, a spacing of 0.5 GHz between diodes allows us to use multiple synapses and reduces neighboring synapses' contributions. This frequency spacing can be converted to a position spacing of roughly $200\mu m$.

3.4.4 . Impedance matching to 50 ohm

Different connection designs can be implemented to create a functioning chain with our spintronic synapses. The standard geometry for a chain is the one presented in section 3.3, where synapses are connected in series. In this case, the voltages from all diodes add up ; however, the resistance of all diodes also adds up. The total resistance then scales with N, the number of synapses. This is an issue for RF power transmission ; indeed, an impedance mismatch with the input power will cause a loss of transmitted power and a non-uniform transmission of the power across the frequency profile, which complicates the prediction of the chain response. We thus need to match the chain to the 50 Ohm source impedance to maximize the output voltage of chains and ensure the same power is transmitted to each synapse independently of its resonance frequency. The best configuration would be to have a load of 50 Ohms no matter the number of diodes, without having to change the shape or material of the diodes.

Another possible architecture emerges when connecting synapses in parallel. Indeed, we demonstrated in section 2.7.3 that synapses made of two diodes in parallel will produce a voltage proportional to the individual responses of the diodes via spin-diode currents summation. This configuration will lead to a resistance scaling in $\frac{1}{N}$, which also results in low transmitted power and a non-uniform frequency profile of this transmission.

To maintain 50 Ohms regardless of the number of diodes, we have developed a hybrid solution in which synapses are partitioned into M blocks in series of P synapses in parallel, thus obtaining a total resistance :

$$R = M \times \frac{R_{synapse}}{P} \tag{3.1}$$

In the case where $M = P = \sqrt{N}$ if N is equal to an integer squared then the total resistance is $R = R_{synapse}$ which can be designed to be 50 Ohms.

The different possible designs are summarised in figure 3.5. In the parallel configuration, all the diodes are connected both to the ground plane and the central line, diodes on each side of the central line have opposite signs and form the pairs needed for each synapse. In the series configuration a synapse block is composed of two diodes in parallel and these synapse blocks are then connected in series by linking the central line to the next ground plane. Similarly the mixed parallel-series design presents blocks of parallel diodes that are connected in series. Figure 3.6 displays an optical image of the mixed configuration design after fabrication.

The aspect ratio to obtain 50 ohm synapses can be extracted from a diode with a length of $20\mu m$ and a width of $80\mu m$. The measured resistance is $R = 12.5$ Ohms. Thus, a diode with an aspect ratio 8 times larger will have a resistance around 100 Ohms. I chose to have diodes with a length of $10\mu m$ and

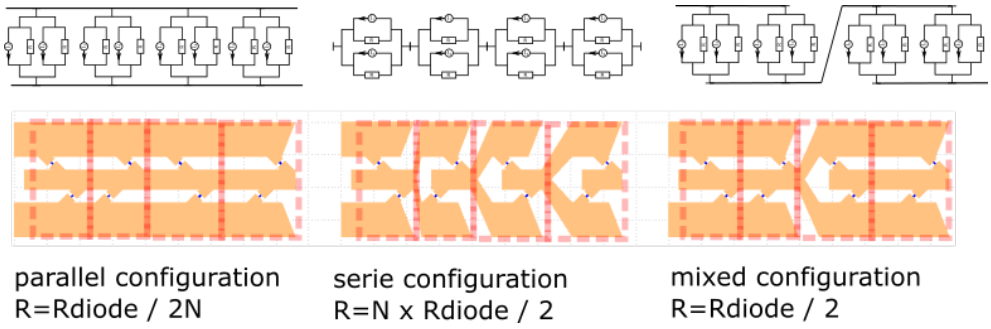


Figure 3.5 – Different chaining configurations for N devices : a) chaining in parallel, where the total resistance scales as $\frac{1}{N}$; b) chaining in series, where the total resistance scales as N ; c) alternate chaining, where the total resistance doesn't depend on the number of devices.

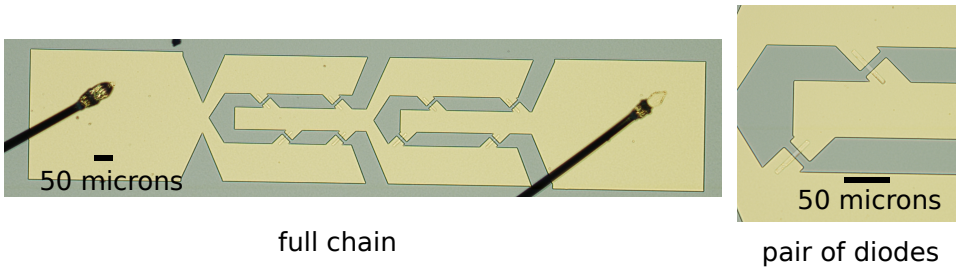


Figure 3.6 – Optical image of chain with mixed configuration.

a width of $5\mu m$. In this case, I obtained pairs with resistances varying from 45 to 65 Ohms. This discrepancy in resistance results from contact resistance between the gold connectors and the Permalloy diodes and variations in the actual width of the diodes after fabrication. The width variation arises because we are working at the resolution limit of the UV lithography equipment.

3.5 . Obtaining Precise Weights in a Chain

Achieving precise weight control in chains of spintronic synapses is a challenge. Several undesired experimental effects should be taken into account. The first issue is that the voltage versus frequency profiles of the diodes in the chain can vary from device to device for two reasons : transmitted power and material discrepancies. The transmitted power varies when the contact resistance is different for diodes in a chain. Additionally, diodes present some discrepancy in shape due to lithography, which can lead to a slightly different resonance frequency versus field profile. The total transmitted power in a chain can also be affected by the wire bonding, which presents non-negligible inductance. Moreover, the magnetic field exhibits some unwanted non-uniform residual fields that can slightly perturb the response of the diodes.

As we will see in the next sections, to tackle these effects as effectively as possible, we have measured three different diodes and averaged their signals to create a reference diode model. The reference diode model is then employed to predict the chain geometries that will produce the desired weights and to determine how the input power should be rescaled to account for the intrinsic decrease in rectified voltage as a function of the diodes' resonance frequencies previously discussed in figure 2.8 a).

3.5.1 . Building a reference diode

To implement chains capable of performing MAC operations, it is essential to precisely understand how an individual diode with any resonance frequency will respond to an input at any given frequency. Therefore, a robust and accurate model for a single diode is required. I decided to base this model on an interpolation of experimental data rather than on the AMR spin-diode voltage theory model in order to take into account all additional effects affecting the experimental signal. The main effect that is included in the interpolation and not in the theory is the variation of the transmitted power in the chain with frequency. This variation is caused by the attenuation of the RF switches, the attenuation from the sample holder RF waveguides, and from the wire bonding, which presents high impedance, thus also impacting the transmission of power with frequency.

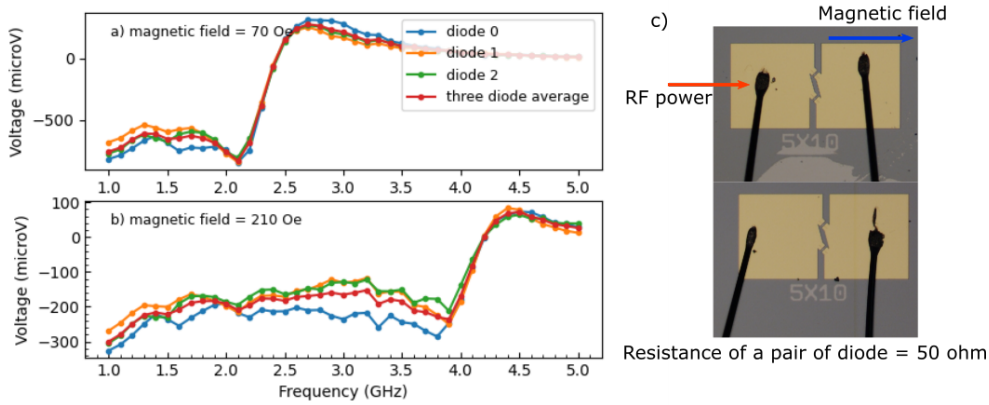


Figure 3.7 – Voltage versus frequency of injected RF power of three identical reference Permalloy spin-diodes of 5x10 microns displayed in c), for magnetic fields of 70 Oe a) and 210 Oe b). The average of these three devices is displayed in red and will serve as the reference signal.

In order to obtain accurate and representative data of the spin-diode voltage, I measured three different diodes and averaged their signals. The spin-diode voltage is recorded by varying the input frequency from 1 to 5 GHz and the external field from -250 Oe to +250 Oe; this scan is done by changing the z position of the devices along the field gradient direction (z). The geometry of the reference devices that I used is displayed in figure 3.7 c). Each device is made of two spin diodes of 5x10 μm in parallel, with the same orientation with respect to the field and hence the same voltage sign. I chose this particular configuration to have a nominal resistance of 50 Ohms which mitigates discrepancies in the transferred power versus frequency that arise from impedance mismatch. To obtain the signal of a single diode voltage from such parallel pairs, we need to divide the measured voltage by 2. This is what we plot in figure 3.7 a) and b) for three different diode devices and for different fields. The averaged signal displayed in red is the one I will use as the reference spin-diode voltage to interpolate versus input frequency and z position. The dependence of the resonance frequency versus z position is also interpolated via this averaged data.

3.5.2 . Prediction of the chain geometry

To obtain a chain that performs a desired weighted sum on multiplexed inputs, we need to ensure that the weights at each input frequency are controlled and match the required values. To demonstrate this, I chose a chain of four synapses addressing four inputs at frequencies 1.75, 2.25, 2.75, and 3.25 GHz. The first requirement for the chain design is that the four synapses that

it contains have their respective resonance frequencies also centered at 1.75, 2.25, 2.75, and 3.25 GHz. This is done by finding the corresponding z positions for each synapse via the reference diode calibration presented above. From these positions, we know the spacing along z between pairs of diodes.

Now that the synapses are placed along z and the frequency profile displays peaks at the input frequencies, we need to tune each of these peaks to have an amplitude that matches the desired weight. The amplitude of each peak is controlled by the shift in z position between the positive and negative diodes of the synapse corresponding to the peak. To predict the z shift, a model representing a double diode synapse is made from the interpolation data of the reference diode :

$$V_{synapse}(f,z) = 0.5 \times (V_{ref}(f, z - \frac{\delta z}{2}) + V_{ref}(f, z + \frac{\delta z}{2})) \tag{3.2}$$

For simplicity of usage, we decide that a given weight should be encoded by the exact same peak amplitude of rectified voltage, whatever the frequency at which the synapse operates. However, as displayed in figure 3.8 a), the amplitude of the rectified voltage peaks decreases strongly with frequency. This implies that to keep a constant peak height, at high frequency, the δz parameter should be much larger than at low frequency. This leads to strong and even unrealistic constraints in diode placement along axis z. If δz's are too big, diodes from different pairs can overlap in space, causing the design to fail. To solve this issue, the injected power is rescaled depending on the input frequency to maintain a constant amplitude of the voltage peak across frequency for a constant δz of 70 μm. The power is rescaled linearly as the decrease of the voltage amplitude can be fitted linearly; see figure 3.8. The injected power is expressed as :

$$P = P_0 \times \frac{-40 \times 3.1(GHz) + 150}{-40 \times f(GHz) + 150} \tag{3.3}$$

Once a set of target weights has been defined, the set of corresponding δz_i parameters is optimized iteratively. Indeed, we want to have the correct weight for each input frequency. This means that we need to have the corresponding desired peak amplitude at each frequency. For this we want to minimize the difference between the terms $V_{chain}(f_i, z_i, \delta z_i)$ and $V_{w=1} \times w_i$ for all frequencies f_i.

In order to exploit the full range of experimentally accessible voltages for encoding the weights, I have set a constraint for the optimization procedure that the maximum rectified voltage in each chain should be equal to V_{range} = $25\mu V$. This is fixed across all chains even if the largest weight is different in

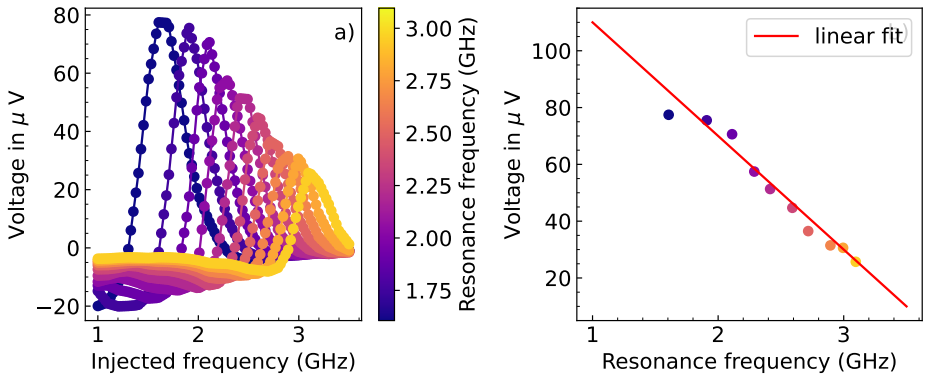


Figure 3.8 – a) Interpolated profile of a synaptic response versus the frequency of RF input while varying the frequency of resonance; the maximum voltage is recorded versus frequency of resonance in b). A linear fit is performed to approximate the decrease in amplitude of the spin-diode voltage with frequency.

each. Therefore, $V_{w=1}$, the value of the output rectified voltage for a weight equal to 1, is different in each chain.

During the optimization process, it is important to take into account the overlap between neighbouring synapses in frequency. δz_i is thus optimized to minimize the difference between the total voltage at frequency f_i, which can be formulated as $(\frac{1}{N}\sum_j V_{synapse}(f_i, z_j) - V_{w=1} \times w_i)^2$. The factor $\frac{1}{N}$ comes from the distribution of the total injected power between the N synapses. This optimization step, which accounts for neighbouring contributions, is repeated 4 times as all δz_i's are updated at each step, modifying the overlaps. The pseudo-code of the optimization procedure is displayed in algorithm 1.

Algorithm 1 Prediction of chain geometry

Require: w_0, w_1, w_2, w_3
  for $k \leftarrow 1$ to 4 **do**
    for $i \leftarrow 1$ to N **do**
      for $j \leftarrow 1$ to N **do**
        $V_{ij} \leftarrow V_{synapse}(f_i, z_j, \delta z_j) \times \frac{P(f_i)}{N}$
      end for
      $V_{neighbours} \leftarrow \sum_{j \neq i} V_{ij}$
      $\delta z_i \leftarrow minimise((V_{synapse}(f_i, z_i, \delta z_i) + V_{neighbours} - V_{w=1} \times w_i)^2)$
    end for
  end for

3.6 . Experimental demonstration of predefined weights

To test the validity of my chain model, I designed three chains, each with different target weights displayed in table 3.1. These three chains have four synapses each processing four inputs at frequencies 1.75, 2.25, 2.75, and 3.25 GHz. According to our interpolation of the response of a single diode versus its position along the z axis, to obtain synapses with frequency peaks centered on these four frequencies, we need to have a spacing of 175 μm, 216 μm, and 245 μm between successive synapses. To obtain the desired weights, our optimization procedure returns a list of four z shifts δz for each chain; the obtained values are summarised in table 3.1. I then fabricated the chains with these predicted δz. After fabrication, I manually optimized the position of the chains along the magnet's z axis to obtain the correct frequency profile.

I have then measured the response of the total rectified voltage in each fabricated chain as a function of frequency. The maximum measured value of each chain V_{chain} did not exactly correspond to the target range $V_{range} = 25\mu V$ that was set during the optimization. I attribute these differences to the wire bonding length that is different in each chain, and different also from the measurements of the reference diode. Indeed, by rescaling the measurements by 1.75, 1, and 1.5, we can see in figure 3.10 that the experimental frequency profiles in blue show a very good matching with the ideal one in red. In order to obtain the experimental weights from these measurements, I then use the normalized values for V_{chain}, presented in the plots of 3.10, that I call V_{chain}^{norm}. Each weight w_i is obtained from the value of V_{chain}^{norm} at frequency f_i, at the maximum target weight in each chain w_i^{max} : $w_i = w_i^{max} \times V_{chain}^{norm}/25\mu V = V_{chain}^{norm}/V_{max}$. Experimental weights are also summarised in table 3.1, and although close to their expected values, they still present differences.

There are several sources of discrepancy. The first one is a misposition of the sample containing the chains. Changing the position can modify the field applied to the chains; the H_z component presents a gradient that is quite uniform in the y position but can vary along x, the direction perpendicular to the sample. A change in the strength of the H_z gradient can affect the spacing in frequency of the fabricated diodes and thus modify the frequency profile. Similarly, an angle in the sample plane can affect the applied field on the diodes. Additional residual components of the field H_x and H_y, while small, present some non-uniformity in space, and a small change of position of the sample compared to reference diodes can translate into the change of these additional components, once again perturbing the frequency response.

Another cause is fabrication variability, which causes the width and contact resistance to vary slightly between diodes. As a result, the currents flowing in diodes of a synaptic pair can be different, leading to a different voltage amplitude generated by the diodes. This variation in shape and contact resistance is visible in the total resistance of the chains, which varies between 50.9 to

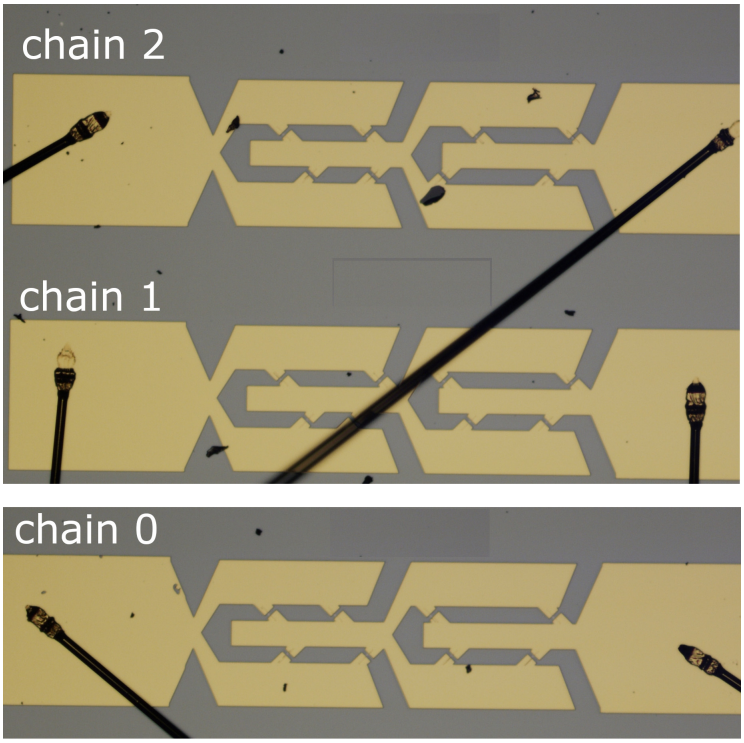


Figure 3.9 – Optical image of the three fabricated four synapses chains

53.2 Ohms. Fabrication can also, due to its imprecision, produce slightly incorrect δz values; however, this effect is, in proportion, much smaller than width imprecision. Indeed, the width is around 5 to 20 times smaller than δz and exhibits a similar lithography imprecision.

Finally, the wire bonding is another cause of imperfect frequency profile. Wire bonds add inductance that breaks the perfect impedance matching. Since the reference diode model has been created with bonded diodes, this effect should be taken into account; however, two bondings can have different inductance depending on the wire lengths and curvatures. This discrepancy in bonding can distort the frequency profile of power transmission and thus the spin-diode frequency profile.

Although not perfect, the experimental profiles obtained demonstrate good enough accuracy to attempt MAC operation, especially in the case of neural networks that are designed to be intrinsically resilient to noise, as will be demonstrated in the next section.

Table 3.1 – Theoretical and experimental weights for 3 different chains of four synapses

	w_0	w_1	w_2	w_3
chain 0 R=51.5 Ohm				
theoretical weights	-0.1322	-0.3949	-0.2745	0.3621
experimental weights	-0.2684	-0.4309	-0.2949	0.3289
$\Delta w(\text{in \% of w=1})$	13.62	3.60	2.04	3.32
$\delta z(\mu m)$	-43	-75	-47	51
chain 1 R=53.2 Ohm				
theoretical weights	-0.2338	0.2530	-0.2336	0.2327
experimental weights	-0.2527	0.2153	-0.141	0.1265
$\Delta w(\text{in \% of w=1})$	1.89	3.77	9.26	10.62
$\delta z(\mu m)$	-62	64	-60	62
chain 2 R=50.9 Ohm				
theoretical weights	-0.2177	-0.1121	0.2409	0.1869
experimental weights	-0.2835	-0.1432	0.2204	0.1573
$\Delta w(\text{in \% of w=1})$	6.58	3.11	2.05	2.96
$\delta z(\mu m)$	-63	-11	82	54

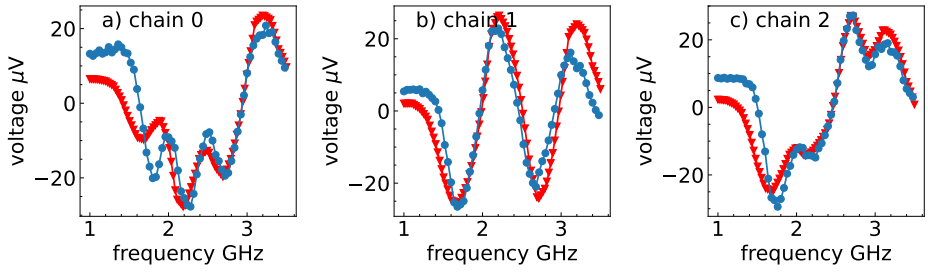


Figure 3.10 – Predicted profile from reference diode interpolation is shown in red and the rescaled experimental measured profile for fabricated samples is shown in blue. a, b, and c correspond to three different sets of weights. Rescaling factors are 1.75, 1, and 1.5.

3.7 . Conclusion

In this chapter, we demonstrated that spintronic synapses made by shifting two opposing diodes can be connected in a chain to implement a multiply and accumulate operation. These chains were designed with blocks of synapses wired in parallel ; these blocks, connected in series, allow the total resistance of the chain to remain close to 50 Ohms. This approach presents two advantages : first, through impedance matching to the source and cables, it maximizes the transferred RF power ; the other advantage is that the total resistance of the chain is identical to the resistance of the calibration device, ensuring a similar frequency response. A calibration device made of two diodes adding up in parallel has been designed, and its spin diode voltage was measured while varying the z position and input frequency. This reference voltage enables us to predict accurately the frequency response of a chain made of several synapses and thus to infer the δz shifts that each synapse needs to implement the correct weight. I tested this optimization procedure by fabricating three chains with predicted δz ; the experimental profiles match the ideal ones within a margin of 5.2$\%$ of the $w = 1$ references.

In the next chapter, we will explore the implementation of a hardware convolutional layer with such chains of spintronic synapses.

4 - RF convolutionnal network on FashionMNIST

4.1 . Summary

The goal of this chapter is to demonstrate a hybrid hardware-software convolutional neural network comprising a hardware, spintronic convolutional layer, and to test this network on a complex task : FashionMNIST [82]. The network is made up of a convolutional layer with three 2x2 kernels, ReLU neurons, and a final fully connected layer. This network is first trained purely in software. This training includes a random noise applied at the output of the convolutional layer to simulate experimental noise; a procedure that helps the network to become noise-resilient. Once the network is trained, the optimal weights for the convolutional layer are extracted, and three chains of four spintronic synapses are fabricated to implement the twelve weights from the three kernels. The convolutional layer requires the input images to be sent as RF inputs. Each image is decomposed into kernel-size blocks that are converted to RF inputs and sent to the three chains to produce an output pixel each. This sliding-kernel operation is performed sequentially until the full input image has been completely screened. To generate the required frequency-multiplexed RF inputs, we fabricated a custom multi-channel RF source. This source is based on crystal oscillators that output an RF signal tunable in frequency; these oscillators are combined with amplifiers to control the output power in each channel. This source requires precise calibration to ensure that the delivered inputs are correct. The position of the sample containing the spintronic chains is optimized and corrected through the evaluation procedure to compensate for its position drift. We obtained very satisfactory results; on the first 100 images of the test dataset, the experimental accuracy reaches 88%, comparable to the software with noise accuracy of 88.4% and slightly lower than the software without noise accuracy of 90%.

4.2 . Introduction

In this chapter, I present a hybrid software-hardware convolutional network. This network is designed to perform image classification on the FashionMNIST dataset [82]. Since our synaptic weights cannot be tuned after fabrication, the network is first trained in pure software. To obtain a network resilient to experimental noise, random Gaussian noise is injected after the convolutional layer; this approach is called noise-aware training. The weights obtained in software are then used to predict the geometry of three spin-

Table 4.1 – Label to clothe type

Label	Clothing type
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

tronic chains that will optimally perform the convolution operation. After the fabrication of these chains, the inference is performed using experimental measurements.

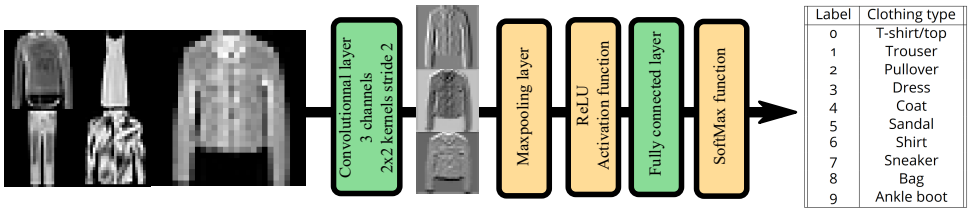
In this chapter, I will introduce the task and the chosen network architecture to solve it, followed by a discussion on how the convolutional part of this architecture can be implemented in hardware. I detail the realization of a custom multi-channel RF source made with commercial elements and its characterization. The noise-aware training procedure will be described and compared to the measured experimental noise. Finally, I show that the experimental network can perform as well as a noisy software network, reaching an accuracy of 88%.

4.3 . Network Architecture

4.3.1 . Task, model and training

Our goal is to solve an image recognition task with an hybrid hardware-software network. The chosen task is to label images of ten types of clothing from the FashionMNIST dataset [82]; the different types of clothing and associated labels are displayed in table 4.1. These images are in gray scale and have a format of 28×28 pixels. The training dataset consists of 60,000 images and the testing dataset consists of 10,000 images.

The neural network chosen to solve this task consists of a convolutional layer with three 2×2 filters, including padding. Padding in convolutions refers to the addition of extra pixels, usually zeros, around the input image's border to control the spatial dimensions of the output feature map. The kernel is slid



Label	Clothing type
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Figure 4.1 – Architecture of the hybrid hardware-software convolutional network : The network is made up of a hardware convolutional layer with three 2x2 kernels and a stride of 1, a software ReLU neuron layer, and a software fully connected layer. The biases of the convolutional layer are also purely software.

by a stride of 1 to produce each output pixel. Then, a max pooling layer is applied with a kernel size and stride of 2. This operation looks at the pixels on which its kernel is applied and keeps only the highest value; in our case, this operation divides the image size by two. These outputs are fed to a layer of ReLU neurons and then passed through a fully connected layer. Finally, a softmax function is applied to generate the probabilities of being in each of the ten classes.

The network is trained purely in software for 10 epochs with a learning rate of 0.01 and using the Cross Entropy Loss. The network achieves 87.63% accuracy on the test dataset. Details about the training procedure, and the introduction of noise in the software model to account for device imperfections are given in Section 4.6.1 : Noise aware training. Once the training is finished in software, we implement in hardware the learned weights of the first convolutional layer. These weights are the coefficients of the kernels of the convolutional layer. This represents 3 times 2×2 coefficients, so a total of 12 parameters to be implemented in hardware. All the operations performed after the convolutional layer are implemented in software. This hybrid hardware-software architecture is represented in Figure 4.1.

4.3.2 . Spintronic hardware convolutional layer

The convolutional layer of the network is made up of three different filters of size 2x2. This layer produces three output images, one for each filter, from the input image; each output image has the same size as the input image since the stride of the kernel is one. Each output image can be expressed as the result of Equation 4.1, where the input image is split into a collection of overlapping regions of the size of the kernels, 2x2 pixels, k is the filter index, and g and h are the pixel height and width indices. As can be seen, Equation 4.1 corresponds to a weighted sum with weight w_{ij}^{k} applied to all 2x2 pixel regions of the input image. This means that the convolutional layer operation can be

implemented with the four-synapse chains presented in the previous chapter.

$$V^k_{out,gh} = \sum_{i=0}^{2}\sum_{j=0}^{2} w^k_{ij} P_{g+i,h+j} \tag{4.1}$$

The spintronic implementation of this convolution operation is displayed in figure 4.3. Each 2x2 region is flattened into a 4 pixel vector. Each pixel of this vector is converted to an RF signal, which frequency encodes the pixel location. For instance the pixel in position (0,0) under the kernel is transformed to an RF signal at frequency $f_0 = 1.75GHz$ in order to have the weight $w_{0,0}$ applied by the chain . This link between kernel weights and frequency address is displayed in figure 4.2. The power of the RF signal encodes the pixel value. The pixel values range from 0 to 255 ; a pixel value of 0 will be converted to an input power of 0 mW, a pixel value of 255 will be converted to the maximum power at the input frequency :

$$P_{max} = 40mW \times \frac{-40 \times 3.1(GHz) + 150}{-40 \times f_{input}(GHz) + 150} \tag{4.2}$$

Similar to the one defined in Eq. 3.3 of the previous chapter, this scaling is implemented to compensate for the difference in the response of the spin-diodes at the four input frequencies. Since, as we have seen, diodes have a lower voltage response at high frequencies, more power needs to be injected to obtain equivalent values. The empirical scaling is extracted from Figure 3.8. The four RF inputs at the four input frequencies are then combined with a power combiner. This frequency-multiplexed signal is then sent to three different chains of four synapses, each implementing a given convolutional filter. These three chains apply different weights to the same four inputs. The total voltages of these chains need to be converted back from voltage values to pixel values to form the three output images. This is done first by rescaling the measured voltage values in each chain by normalizing voltages corresponding to a weight equal to 1 : $V_{max} = \frac{V_{range}}{w_i^{max}} = 63.3, 98.8, 103.8, \mu V$ in order to match the range of weights that they each encode (see section 3.6 for the detailed procedure), then by multiplying this value by 255. Ideally, the three chains need to be placed at the same z-position in the field gradient to ensure the same list of resonance frequencies in the three chains. In practice, due to space constraints on the sample design, chain 2 is not aligned with chains 0 and 1, as can be seen in the optical image in Figure 4.3.

To produce the full output images, the input image needs to be completely scanned by the filters. Thus, after producing an output pixel with each filter, the kernels are slid with a stride of 1 to process the next 2x2 portion of the input image.

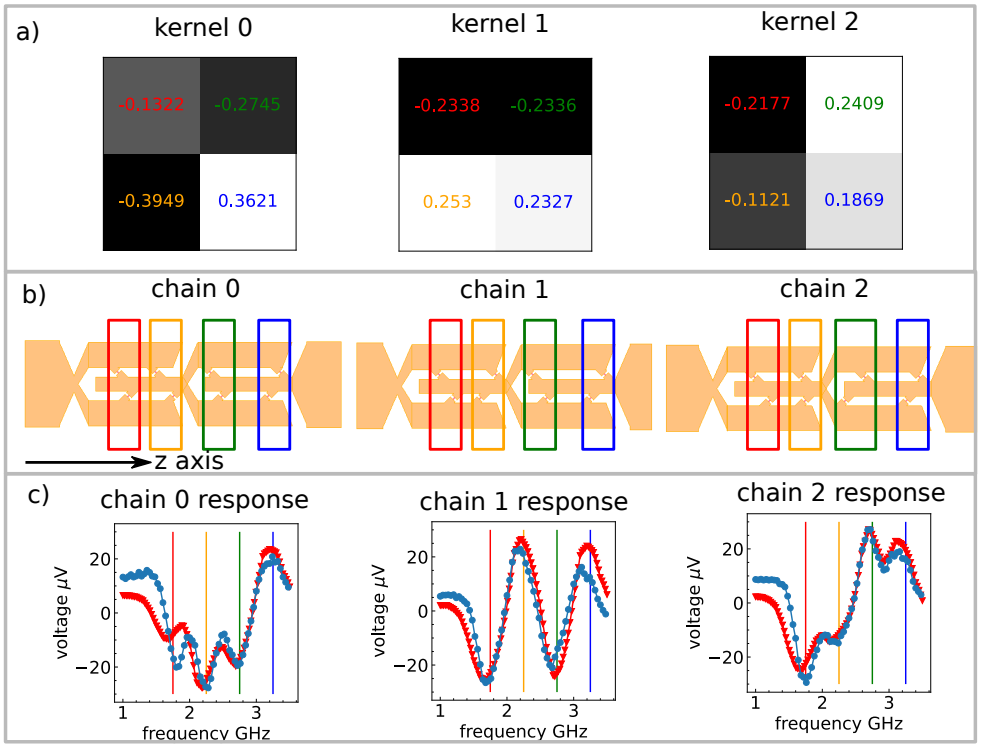


Figure 4.2 – a) Obtained kernels after training, kernel 0 corresponds to a diagonal lines filter. Kernel 1 is close to a horizontal lines filter and kernel 2 to a vertical lines filter. b) Geometry of the chains implementing such kernels, synapses are highlighted in colors matching the kernel weight they implement. c) Spin diode voltage response of each chain, expected one in red and experimental one in blue. The colored vertical lines display the input frequency address of each weights implemented in the chains.

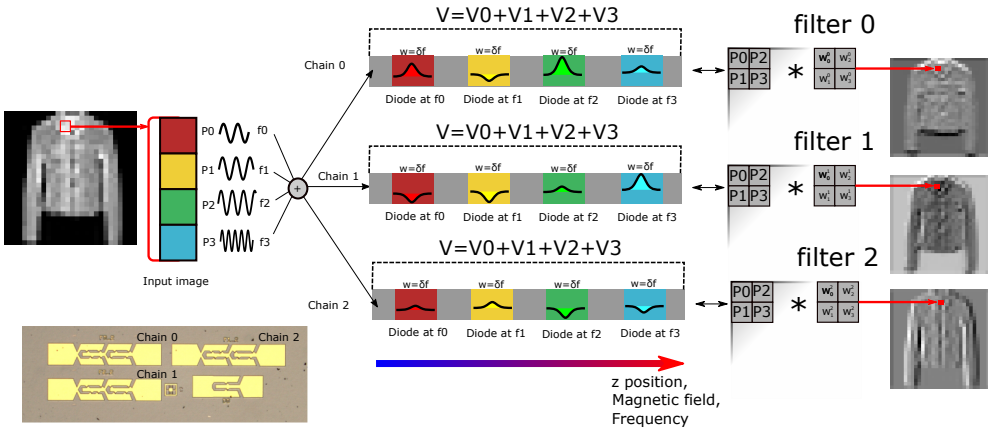


Figure 4.3 – Hardware architecture of the hardware convolutional layer made up of three chains, each one playing the role of a convolutional filter.

4.4 . Experimental setup

The experimental setup is composed of three principal parts : the production of RF inputs, the voltage measurement setup, and the control of the magnetic field applied to the sample.

The inputs are generated with a custom RF source that is described in Section 4.5. This source delivers multiplexed inputs controlled in power and frequency. Two RF switches are placed after this source. The first one allows choosing where the RF input comes from : my custom multichannel RF source or a standard high-precision mono-channel RF source. This last source is the one with which I performed previous calibrations and measurements, as it has well-controlled properties and very low noise in power and frequency. The second switch can redirect the input signal to four different ports. Three of these ports direct the signal to the three chains of the convolutional layer. The fourth port directs the signal to a power spectrum analyzer ; this instrument is used to calibrate the custom RF source in power and frequency.

The voltage measurement section is composed of a multi-channel nanovoltmeter and a bias T. When measuring the voltage response of a spin-diode, the AC component of the input RF needs to be discarded to only measure the DC part. To do so, samples are connected to the inputs of the nanovoltmeter via a bias T. The high-inductance branch is connected to the nanovoltmeter to collect only DC components, and the high-capacitance branch is connected to the RF source to pass only AC input signals. The nanovoltmeter can be connected to three samples at the same time and can switch between them.

Finally, the sample is mounted on a sample holder that can be moved in the two horizontal directions, x and y, with controllable Thorlabs motors.

This allows precise placement of the sample between the array of 4 magnets creating a field gradient. This array of 4 magnets can be moved vertically to complete the full 3D placement of the sample in the magnetic field. Additionally, this array of magnets can be rotated in a small range of $\pm 10^{\circ}$ around the vertical axis and can be fully rotated around the y-axis to compensate for any misalignment of the sample with the field direction. All these components are controlled directly with Python.

4.5 . Multi-input source

4.5.1 . Description

As described in the previous section, the convolutional hardware layer operates on four RF inputs with different frequencies. However, multi-frequency RF sources are very specialized and expensive equipment. Given that our application does not have the stringent constraints on frequency and phase stability associated with other RF applications, we decided to build a custom multi-channel RF source using basic commercial elements offering lower performance at a price range of less than 1k€, compared to high-end RF equipment such as Arbitrary Waveform Generators, which can quickly approach 100k€ in price.

The source design and implementation are displayed in Figure 4.4. This source is made of 4 individual blocks that can produce an input controlled in frequency and power. Each block is made with the same structure : a voltage-controlled oscillator from the Crystek company, model CRBV55BE, highlighted in red in Figure 4.4 a), is connected to a low-noise RF amplifier, represented in orange. This electronic oscillator outputs an RF signal whose frequency can be tuned over a broad range, from 1.53 to 2.9 GHz, by changing a control voltage from 0 to 20V. The emitted power can then be tuned in amplitude by adjusting the voltage supplying the amplifier ; reducing the supply voltage lowers the effective compression point and gain. If the amplifier receives no voltage as input, it doesn't let the RF signal pass, and by gradually increasing the amplifier supply voltage, the amount of transmitted power increases. The voltage supplied to the amplifier and the crystal oscillators is controlled via computerized voltage sources.

The outputs of these four individual blocks, controlled in power and frequency, are then combined via a power combiner. The resulting mixed signal is then re-amplified as the power combiner presents a strong attenuation of 10 dB. We want to deliver the maximum power to the chains to maximize the spin-diode voltage compared to the measurement noise. The limitation to the input power that we can send at a given frequency is linked to the compression point of the final amplifier. Above a given value of input power, the

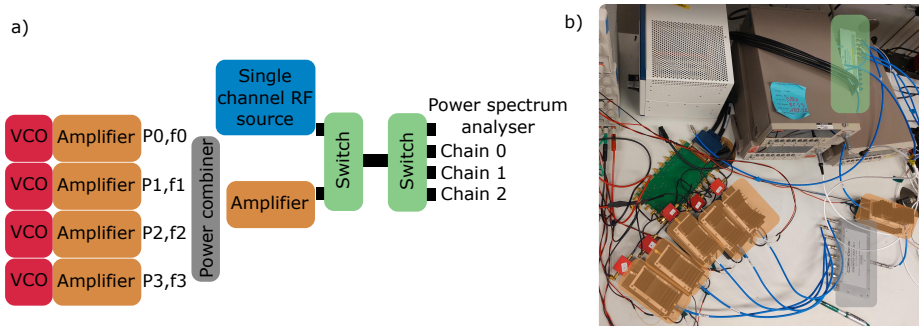


Figure 4.4 – The multi-channel source is made up of four basic units, each comprising a voltage-controlled oscillator that has a tunable GHz frequency and an RF amplifier with a voltage-controlled gain. The powers from these units are combined and re-amplified before being distributed to the chains or a power spectrum analyzer.

gain of the amplifier starts to decrease. This is an issue in the case of a multi-frequency input as the compression point depends on the total input power comprising all frequencies but also depends on the frequencies present in the input. Thus, when sending several powers at different frequencies with a total near the global compression point, the ratio between each power will be modified. Consequently, to avoid this kind of non-linear mixing effect of powers, we need to stay well under the lowest compression point for our four input frequencies. The latter being 100 mW, I chose a maximum power of 40.0 mW at 3.25 GHz and thus 20.0 mW, 13.3 mW, and 10.0 mW for the three other frequencies. This is a good trade-off because, even when all inputs are at their maximum value, the total injected power is 83.3 mW, which is below the compression point. I added attenuators at the output of each RF channel in order to match as well as possible this maximum needed power and the experimental maximum RF power delivered by the amplifiers. This attenuation is 0 dB for a frequency of 3.25 GHz, 6 dB for 2.75 GHz, 10 dB for 2.25 GHz, and 13 dB for 1.75 GHz.

4.5.2 . Frequency and power calibration

In order to have a well controlled and precise input in frequency and power from my multi channel source a calibration step is needed. We scanned the output power and frequency of the four channels from the custom source while varying the voltages controlling frequency of the crystal oscillator and controlling the gain of the amplifier. The results of this calibration are displayed in figure 4.5 a) and b). In graph a) the dashed lines indicate the functioning points of the four crystal oscillators. Oscillators 0, 1 and 2 are the same model number : CRBV55BE and operate at 1.75, 2.25, 2.75GHz respectively,

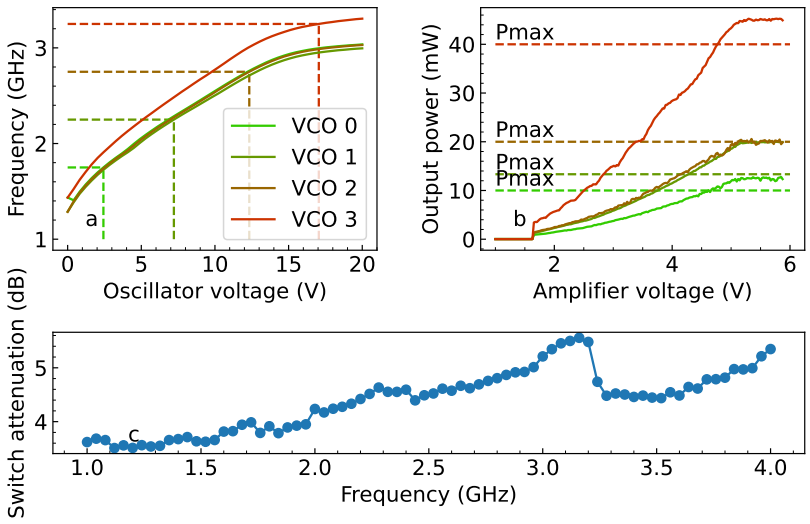


Figure 4.5 – a) Frequency of output RF power versus tuning voltage for four different oscillators, the chosen functioning points are represented by the intersection of dashed lines. b) Output RF power versus amplifier voltage for the four oscillators at their functioning points. The maximum power required at each frequency is represented in dashed lines, attenuators have been added after amplifiers to use the broadest range of voltage possible. c) Attenuation versus frequency profile for the RF switches.

while a different model : CRBV55CW had to be used for oscillator number 3 to reach a higher frequency : 3.25GHz. The required tuning voltages to reach the correct frequencies span between 2.5 to 17.5V.

Figure b) presents the curves of output power versus amplifier voltage for all four channels after adding the necessary attenuators. The maximum needed output power for each input frequency, $P_{max} = 40, \text{mW} \times \frac{-40 \times 3.1, (\text{GHz}) + 150}{-40 \times f_{input}, (\text{GHz}) + 150}$, is represented for each oscillator with a dashed line. We observe that the output power is quite linear with amplifier voltage in a range of 1.7 to 5V. There is a threshold value to output RF power ; the corresponding voltage is 1.7V, and this threshold is accompanied by a jump from 0 mW of output power to a non-zero value between 1 to 5 mW. Additionally, above a voltage of 5V, the output power doesn't increase anymore. The attenuators that fix the maximum output power to the desired values also reduce the height of the RF power jump, as the low threshold voltage is reduced, allowing for a more accurate range of output powers. Moreover, the range of voltages to use is larger, thus improving the resolution in mW/V.

The output RF powers are measured after the RF switches with a power spectrum analyzer. However, the single-channel source has been used to calibrate the spin diodes; the considered power was the one delivered by the source, thus before the RF switches. The solution to convert the powers measured after the switches is to take into account the attenuation profile by frequency of the switches. To measure this attenuation, an RF signal with known power is sent through the switch from a precise RF source; the signal at the output of the switch is then measured with a power spectrum analyzer. This measurement is repeated, covering the full input frequency range of 1.75 to 3.25 GHz. This profile is displayed in Figure 4.5 c). From this data, we can calculate the power before the switches from the values after : $P_{before}(\text{dBm}) = P_{after}(\text{dBm}) \times 10^{\frac{A(f)(\text{dB})}{10}}$, with A being the attenuation from the switches. The powers displayed in graph b) are these corrected powers before the switches.

4.6 . Weight prediction for a Noise resilient Network

Our neural network cannot be trained in hardware directly since the synaptic weights are written via lithography. Consequently, the network is first trained purely in software to then extract the synaptic weights of the convolutional layer and implement them on a sample. At this point of development, we didn't know exactly what the noise from the experimental setup would be, as our custom RF source was not fully built, but we knew that we needed to anticipate it. Thus, the software network is trained with noise-aware training. This technique consists of adding random noise during the training to obtain weights resilient to noise. After fabrication of the final sample, we characterized the noise on the RF inputs from our source and on the voltage output to ensure that our noise choice was reasonable. These results are displayed in the next section.

4.6.1 . Noise aware training

In order to build a network resilient to noise, we can use a technique called noise-aware training. This consists of injecting a given noise in the forward pass of the training. A new random value of noise is generated at each forward pass. Training a network with this imposed constraint forces the network to find weights resilient to the chosen noise type. In this study, we chose to add Gaussian noise to the output of the convolutional layer; this Gaussian noise is proportional to this output, see Equation 4.3. The noise level is controlled by the parameter n_l. In this training procedure, I chose to set it to 0.5, corresponding to a noise level of 50% of the convolutional layer output.

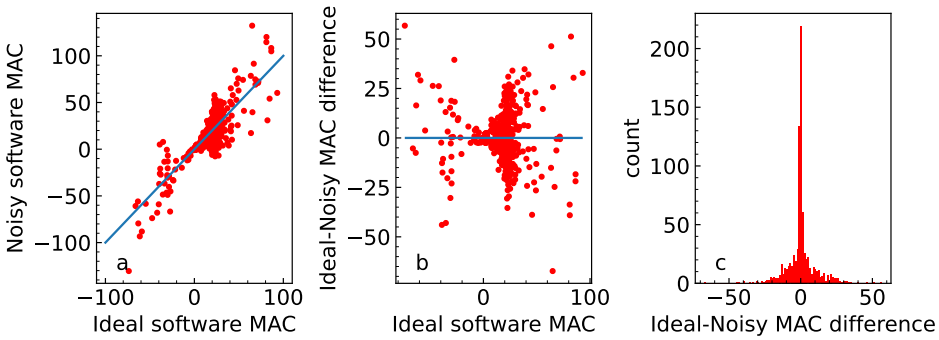


Figure 4.6 – a) Noisy versus perfect multiply and accumulate in the case of the chosen noise for noise-aware training, data is displayed for a single image and a single filter. b) MAC error : noisy minus perfect MAC. c) Distribution of the MAC error.

$$NoisyOutput = Output \times (1 + n_l \times N(0,1)) \tag{4.3}$$

This noise structure was designed before characterizing the multi-channel RF source, assuming that the noise on the output of the convolutional layer is mostly proportional to the input value and can thus be estimated as proportional to the convolution output. As we will demonstrate in Section 4.9, the noise is actually more of the addition of systematic biases in the weights and a random Gaussian noise uncorrelated to the output value.

Figure 4.6 displays the noisy output of the convolutional layer for the chosen noise level of 50%. Graph a) displays the noisy experimental output of the multiply and accumulate operation versus its non-noisy version. In graphs b) and c), the difference between these two quantities is displayed as a function of the output and in the form of a histogram. We see, as expected, that the noise presents an hourglass shape centered around 0. Most of the errors are very small; this especially comes from the background of the FashionMNIST images. These background pixels have zero value and will lead to null output with null noise.

The training results show that our noise aware training is efficient, the test accuracy without noise is 87.63% and with 50% noise is 86.34%. The loss of accuracy is only of 1.3% which is quite small. The final predicted weights and δz parameters are the one summarised in the previous chapter in table 3.1.

4.6.2 . Evaluation of the experimental noise

When implementing a software network in hardware, we introduce experimental sources of noise. In order to build a network resilient to these noises, we need to characterize the noise sources, starting with the noise on the inputs and the noise on the measurement of the outputs of the hardware layer. I measured the noise from the nanovoltmeter measuring the output voltages and noise in frequency and power from the multi-channel source. Results are displayed in Figure 4.7. In graph a), the level of noise is displayed for the input power delivered by the multi-source at different frequencies and values of power. The level of noise in power doesn't exceed 0.5$\%$ for all frequencies of interest and tends to scale down with the absolute value of delivered power. This level of noise is low and should not be an issue for sending correct inputs to the hardware layer. In graph b), the noise levels on the frequency of the RF inputs at different frequencies and power are displayed. This noise level is even smaller, with a maximum value of 0.005$\%$, and also tends to scale down with the actual RF output power. This noise corresponds to an error of around 100 kHz for inputs around 2 GHz, which is negligible.

Finally the measurement noise from the nanovoltmeter is displayed in graph c). This noise doesn't depend on the value of voltage measurement and has a constant value that can be estimated to be around 20nV. The value of the maximum voltage peak for our three previously fabricated chains was $V_{range} = 25\mu V$, this result has been produced with a base RF power of 200mW, as we use here a base power of 40mW, thus the voltage peak value can be approximated to $5\mu V$. In this case the noise level is 0.4%, it's equivalent to the noise on the input. It could be reduced by averaging voltmeter measurements, but the spin-diode measurement requires to switch on and off the input, and this is time-consuming. In this study voltages are not averaged to keep a high measurement speed.

These two sources of noise are quite small compared to the noise in the noise aware training that was considered to be 50% of the output.

4.7 . Position optimisation inside the field gradient

As seen in the previous chapter, the experimental frequency profiles present some discrepancies with the expected ones. To implement the most correct weights, we can fine-tune two parameters. The first one is the reference voltage V_{max} for each chain. Since the wire bonding differs in length and curvature for our chain compared to the reference devices, and the resistance of the chain can also be slightly different due to lithography, the proportion of transmitted power can vary, leading to a different strength of the spin-diode signal. Rescaling V_{max} for each chain can compensate for this issue of power

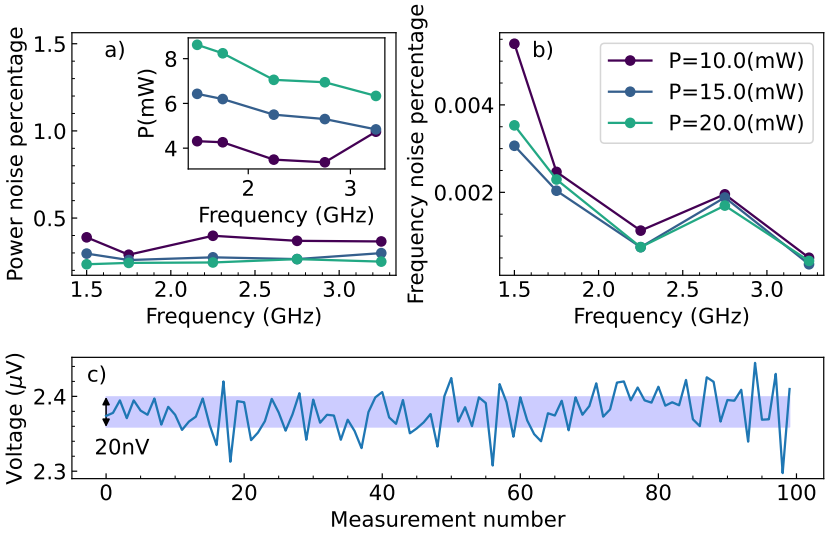


Figure 4.7 – a) Percentage of noise on the output power of the multi channel source for different frequencies and power, the noise is determined as the standard deviation for 100 points, inset represents the actual output power. b) Percentage of noise on the output frequency of the multi channel source for different frequencies and power, the noise is determined as the standard deviation for 100 points. c) Measurement error on the nanovoltmeter.

transfer. The second parameter that should be optimized is the position of each chain inside the field gradient. We need to place the fabricated chains at the exact positions to achieve the correct magnetic field on each diode. This is complicated due to the fact that samples are glued manually with a precision that cannot be lower than 1 mm, while we need micrometric resolution to obtain the correct profile, especially along the gradient axis z.

To find the best parameter z for each chain and the associated reference voltage, I implemented an optimization procedure based on Optuna, a Bayesian optimization Python package that is described more in depth in Section 6.6. The optimization procedure is summarized in Algorithm 2. I designed a metric M, visible in Equation 4.4, to minimize in order to best match the weights extracted from the noise-aware training. This metric evaluates the difference between the experimental voltage obtained by sending four inputs P_i and the expected voltage coming from the theoretical weights applied to these same inputs. This difference is calculated for 100 sets of four inputs, extracted to be representative of the FashionMNIST dataset. Here, the background of images is excluded as they all present inputs equal to zero.

$$M = \frac{1}{100}\sqrt{\sum_{j=0}^{99}(V_{exp,j} - \sum_{i=0}^{2} V_{max} \times w_i \times P_{i,j})^2} \tag{4.4}$$

The experimental reference voltage V_{max} is computed to minimize the optimization metric M. Optuna is run on 25 trials with different z positions for each chain to try to find the optimal z to best minimize the metric M. After this optimization step, I retrieve three values of optimal V_{max} and three values of optimal z, one for each chain.

This calibration procedure is applied before performing inference on the FashionMNIST images. However, the spin-diode signals tend to drift over time. The causes of this drift are complicated to identify. One of them is the position drift; since the z position is different for each chain, we need to move the sample holder along the z axis. Due to the RF cable rigidity and fixation of the sample holder with plastic screws, moving the sample along the z axis can also cause a drift along the z, y, and x positions. Another possible source of drift can be the modification of the contact resistance over time due to oxidation mechanisms and heating created by the RF injected power. To compensate for these drifts, the optimization procedure is performed every ten input images. It is important to note that performing a convolution with three filters on an input image takes around one hour due to the necessity to wait for the stabilization of the custom RF source.

In Figure 4.8 a), the optimization metric is displayed while sending input images; the optimization steps are represented by dashed lines. We see that

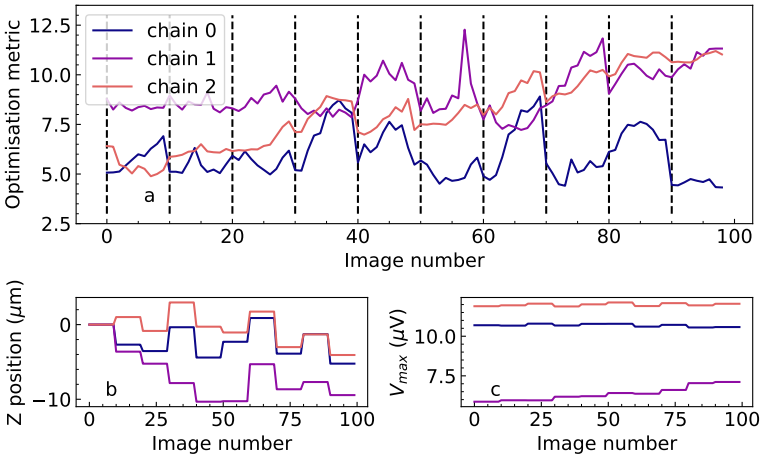


Figure 4.8 – a) Optimization metric displayed for the three chains across the inference for 100 images. The optimization procedure on the sample z position and reference voltage is performed every 10 images; values for these two quantities are displayed in b) and c).

between optimization steps, the optimization metrics tend to drift to higher values for all chains. When performing an optimization step, the optimization metric is reduced. In Figure 4.8 b) and c), the optimal z positions and optimal reference voltages for the first one hundred images are displayed. The z positions vary by an amount of a few microns, up to 10, while the reference voltages are quite stable through optimization.

Algorithm 2 Optimisation procedure for sample position

Require: w_0, w_1, w_2, w_3
  for $k \leftarrow 1$ to 25 **do**
    for $g \leftarrow 1$ to 3 **do**
      $z_g \leftarrow z_{trial}$
      $M \leftarrow \frac{1}{100}\sqrt{\sum_{j=0}^{99}(V_{exp,j} - \sum_{i=0}^{2} V_{max,g} \times w_i \times P_{i,j})^2}$
      $V_{max,g} \leftarrow minimise(M(V_{max,g}))$
    end for
  end for

4.8 . Summary of the experimental procedure for FashionMNIST

The full procedure to process images from the test dataset is the following. The sample is aligned for the first time to ensure correct weights. Then the first image is processed. The first 2x2 area of this image is converted to a set of 4 inputs at the 4 working frequencies and sent to the sample. The voltages from the three chains are recorded and converted to pixel values to fill the three output feature maps. The kernels are then slid by 1 to cover a new area of 2x2 pixels. These values give new inputs, and the voltage measurements are performed once more to produce new output pixels. These steps are repeated until the full input image has been covered. Once all three output maps have been generated, they are fed to the rest of the network, and a label is predicted for the input image. This full classification operation is then repeated for the next image, and so on. Every ten images, a new alignment is performed. This full procedure is displayed in Algorithm 3.

Algorithm 3 Classification of FashionMNIST images

for $step \leftarrow 0$ to 100 **do**
  $Im \leftarrow dataset(step)$
  if mod(step)=10 **then** optimise z positions
  end if
  for $i \leftarrow 0$ to 28 **do**
    for $j \leftarrow 0$ to 28 **do**
      for $k_0 \leftarrow 0$ to 1 **do**
        for $k_1 \leftarrow 0$ to 1 **do**
          $RFinputs(freq_{(k_0+k_1)}) \leftarrow ToPower(Im_{i+k_0,j+k_1})$
        end for
      end for
      for $channel \leftarrow 0$ to 2 **do**
        $PosZ \leftarrow OptimalZ_{channel}$
        $Voltage_{channel} \leftarrow Measure_{channel}$
        $OutIm_{i,j,channel} \leftarrow ToPixel(Voltage_{channel})$
      end for
    end for
  end for
  $Label \leftarrow SoftNetwork(OutIm)$
end for

4.9 . FashionMNIST results

I tested the hybrid hardware-software network on the first 100 images of the FashionMNIST dataset. Figure 4.9 displays different comparisons between the experimental results of the convolutional layer and the theoretical expectations. In graph a), the experimental output of the convolutional layer for the three filters is displayed versus the ideal software output. We notice that the level of noise is very low compared to the noise level injected in the noise-aware training. Our experimental noise is therefore likely not to perturb the classification too much. In graph b), the difference between the experimental and theoretical output is represented versus the theoretical output. This difference corresponds to the experimental error. It is important to note that this error doesn't correspond to Gaussian noise centered around zero. The error corresponds more to systematic biases, mostly due to systematic errors in weights values. We can confirm this by looking at the distribution of errors for the backgrounds of images (pixel values = 0) and their cores.

In figures c), d), and e), the error distribution for the three different chains is displayed. The error on the background of the image is in light color, and the core of the image is in darker color. The background corresponds to all inputs being equal to zero ; from the corresponding error distribution, we can visualize only the nanovoltmeter noise. This error appears as Gaussian noise centered on zero. The error from the core of the image presents a different shape, especially for filters 0 and 2, where the errors are not centered on zero. This highlights the role of weights, which in this case are slightly too positive, thus shifting the error distributions to negative values. This systematic error seems to be the main factor to reduce to avoid misclassification ; however, it is caused by lithography imprecision and cannot be directly corrected.

Now that the experimental noise on the convolutions are well known we can focus on the classification results. The experimental accuracy on the first 100 images is 88%, the software accuracy with the gaussian noise used for noise-aware training, on these 100 first images is 88.4%, and without noise it is 90%. The software accuracy with noise is calculated by passing each of the one hundred input images 1000 times through the network with a different random noise. We achieved performances similar to the noisy software with the experimental network. Thus even if the noise in the noise-aware training procedure was not exactly of the same nature and amplitude as the experimental noise measured afterwards, this choice was close enough to reality to transfer the network in hardware with success. The confusion matrices for the noisy software model and for the experimental model are displayed in figure 4.10 a) and b). We see from this matrices that both networks tend to mix and misclassify pullovers, shirts and coats. This is expected as these types of clothes are very similar, adding only a small noise in the form of a Gaussian

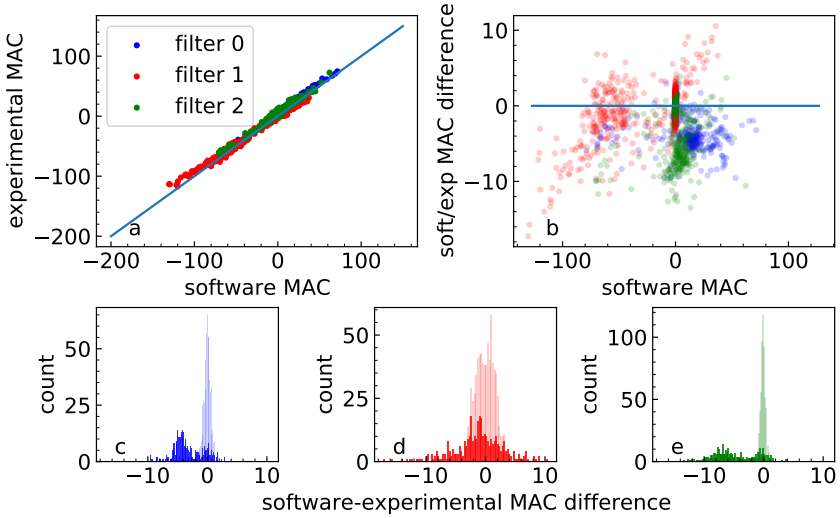


Figure 4.9 – a) Experimental MAC versus software MAC for the three chains receiving four frequency-multiplexed inputs, b) displays the difference between ideal software MAC and experimental one, the discrepancy mainly comes from the experimental error on weight values. c) d) and e) display the distribution of this error in light color for the background of images where input is null, this probes the error due to measurement noise. The error in dark corresponds to the clothes area and highlights the weights error.

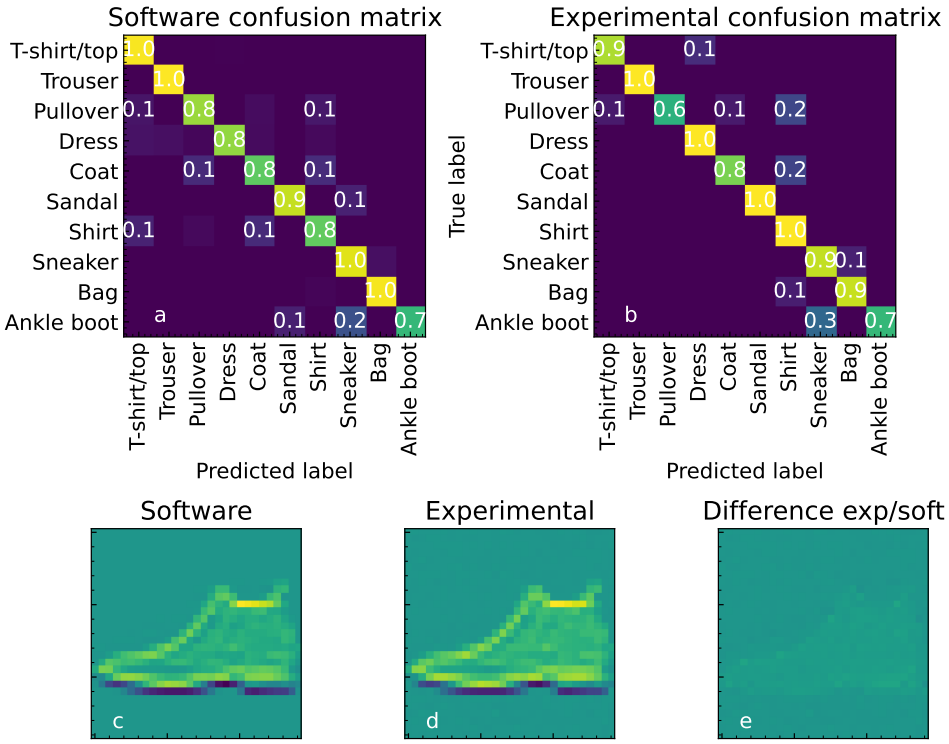


Figure 4.10 – Confusion matrices on the 100 first images of FashionMNIST for a) the software with noise model b) the experimental model. Convolved image for a) the software with noise model b) the experimental model, c) displays the difference between the two images.

noise or systematic bias is sufficient to change the assigned labels of some examples of these clothe types, hence the similar accuracy between noisy software and experimental networks. Figure 4.10(a) and (b) show an example image after convolution in (a) software and (b) hardware. The difference (c) is close to zero, demonstrating the high quality of the hardware convolutions with the RF Permalloy spin-diodes.

4.10 . Conclusion

In this chapter, I proposed a hybrid software-hardware architecture of a convolutional network. The convolutional layer is implemented in hardware with chains of spintronic synapses, with the remaining part of the network, mainly ReLU neurons, max pooling, and a final fully connected layer, being

implemented in software. The network is trained in software, and the obtained weights are then used to predict the correct geometry of the chains of spintronic synapses constituting the hardware part. The training is conducted in the framework of noise-aware training, meaning that I added random noise during training to obtain a network resilient to perturbation.

To test the obtained network, I built a custom multi-channel RF source that I calibrated and characterized. Our experimental noise appears smaller than what was implemented in the noise-aware training, but it is also less similar to a random distribution and more similar to a systematic bias. After testing the network on the first 100 images of the FashionMNIST dataset, I reached an accuracy of 88%, which is comparable to the accuracy of the noisy software network : 88.4%. I had to implement a position optimization procedure applied every ten images to limit the drift of the spin-diode signals. However, this drift could be further limited if the sample was not moved during the inference phase. By fabricating all chains aligned along the z axis, I could perform all my voltage measurements without moving along the z axis. This is a possible path of improvement to reduce the drift in the spin-diode voltage. These results have been accepted at the IEDM 2024 conference and I am a first author of the article that will be published in the proceedings.

5 - Demonstration of a network of spintronic dynamical neurons

5.1 . Summary

In this chapter we demonstrate the possibility to use STNO's natural dynamics to process time varying inputs. We propose a network made of a single layer of coupled STNOs. The devices receive inputs as DC currents and their output is the emitted RF power. These injected DC currents are the parameters on which are applied synaptic weights. The dynamics of the STNOs is simulated through a simplified version of Slavin et al. auto-oscillator model [77]. The resulting network is trained through truncated backpropagation through time to classify time series made of sine and square waves. This task requires both non-linearity and memory. The performance of this trained network is compared to the the performance of the same configuration treated as a reservoir computing system where only the output layer is trained and not the parameters controlling the dynamics of the neurons. The fully trained network presents higher accuracy for a same number of neuron, it can reach 100% accuracy with only two neurons while the reservoir computing approach requires at least 16 neurons to reach the same result.

5.2 . Introduction

The studied network takes inspiration from the article describing liquid time-constants [103] and standard RNNs. We wanted to study the training and inference of a network of dynamical neurons, specifically spintronic neurons for possible future hardware implementations. This kind of neuron helps enrich the complexity of the network by adding a complex and tunable dynamics while keeping a low number of devices.

A static neuron is defined as a non-linear function f applied to an input I; here, we will denote x as its output, thus $x = f(I)$. A dynamical neuron is defined by an internal variable x obeying an ordinary differential equation (ODE) that depends on an external input I and generally also on the variable x itself. This equation can be formulated by Equation 5.1.

$$\frac{dx}{dt} = f(x, I) \tag{5.1}$$

The steady-state value of a dynamical neuron submitted to a constant input is equivalent to the activation function of a static neuron. However, the

dynamic evolution of the output value of a dynamical neuron can have two interesting properties for neuromorphic computing. On one hand, the neuron value depends on previous inputs, giving rise to a memory that can be of crucial importance in time-dependent tasks. On the other hand, the time evolution of the neuron through an ODE gives rise to a richer, more non-linear behavior, which is capable of increasing the complexity of a network with a low number of devices compared to static devices.

5.3 . Spintronic neuron model

According to Slavin et al. [77], a universal auto-oscillator with negative damping can be modeled with a simple mathematical framework. Current-driven spin torque oscillators (STOs) fall into this category. For these spintronic devices, the auto-oscillatory behavior occurs when a current is injected as input, and as this current gets spin-polarized, the magnetization is driven into precession at the device resonance frequency. The magnetization precession creates an oscillating resistance that, combined with the constant current, leads to RF emission. In the following, we will focus on the dynamics of the RF power emission. This power can be modeled with Equation 5.2. In this equation, x is the normalized emitted RF power evolving through an ODE dependent on I, the electric current flowing through the free layer of the device. Γ_G is the Gilbert damping, and Q is a non-linear damping term ; these two damping terms describe the natural relaxation of the oscillator. σ is a physical coefficient that converts the current into a drive of the microwave power.

$$\frac{dx}{dt} = -2(\Gamma_G(1 + Qx)) - \sigma I(1 - x))x \tag{5.2}$$

This model can be simplified to keep only two main terms, the linear relaxation term and the drive applied by an input current. Moreover we can group physical coefficients to simplify the expression, $\sigma = 1GHz/mA$ and $\Gamma_G = \gamma$. The obtained simplified spintronic neuron model is presented in Equation 5.3. Here $I(t)$ is the input fed to the neuron including the σ coefficient, and γ is the characteristic decay.

$$\frac{dx}{dt} = -\gamma x + I(t)x(1 - x) \tag{5.3}$$

This non-linear ODE has several very interesting properties that makes it the perfect candidate for a dynamical neuron. First the neuron output value is naturally bounded between 0 and 1, indeed the input $I(t)$ drives the neuron but as the neuron value approaches one of the two boundaries 0 or 1 either the term x or $(1 - x)$ goes to 0 canceling the drive and thus keeping x above 0

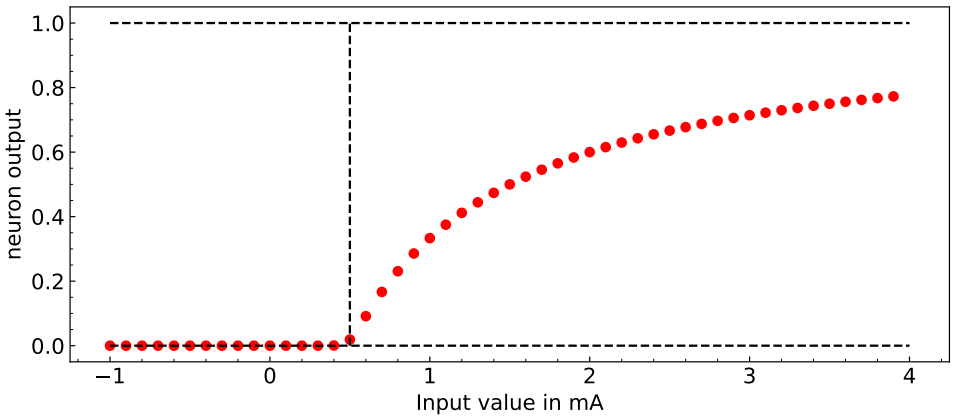


Figure 5.1 – Steady-state profile of a spintronic neuron with $\gamma = 0.5GHz$ versus input value in mA.

and below 1. The fact that $x(t)$ is naturally bounded makes it a natural liquid time-constant network. Second, the drive I(t) can either drive the neuron up or down which is an interesting property compared to the base equation in the first liquid-time article [103] where the decay γ is the only negative term in the derivative and where the neuron cannot decay faster. Here the liquid time constant $-\gamma + I(t)(1 - x)$ can vary between $]-\infty, +\infty[$.

We obtain a steady state profile versus input that has a nice non-linear shape and resemblance to the well-known ReLU function used for many neurons, see figure 5.1. The decay γ defines the activation threshold, here $\gamma = 0.5GHz$.

5.4 . ODE simulation method

Simulating dynamical neurons requires solving ordinary differential equations. Several methods can be employed to solve an ordinary differential equation ; some of them can be analytically solved, but most of them, especially those submitted to time-varying inputs, need numerical solving methods. The simplest solving method that can be proposed is the Euler method, see Equation 5.4. This numerical method consists of computing the time derivative of the dynamic variable x, and then updating the x value by summing its previous value and the derivative $\frac{dx}{dt}$ multiplied by a small time step δt. This method presents several advantages : it's very simple to use, implement, and debug, and requires low computing power, leading to faster operation. The main drawback is its low accuracy ; however, we don't require rigorously exact differential computing results as in physical device simulations.

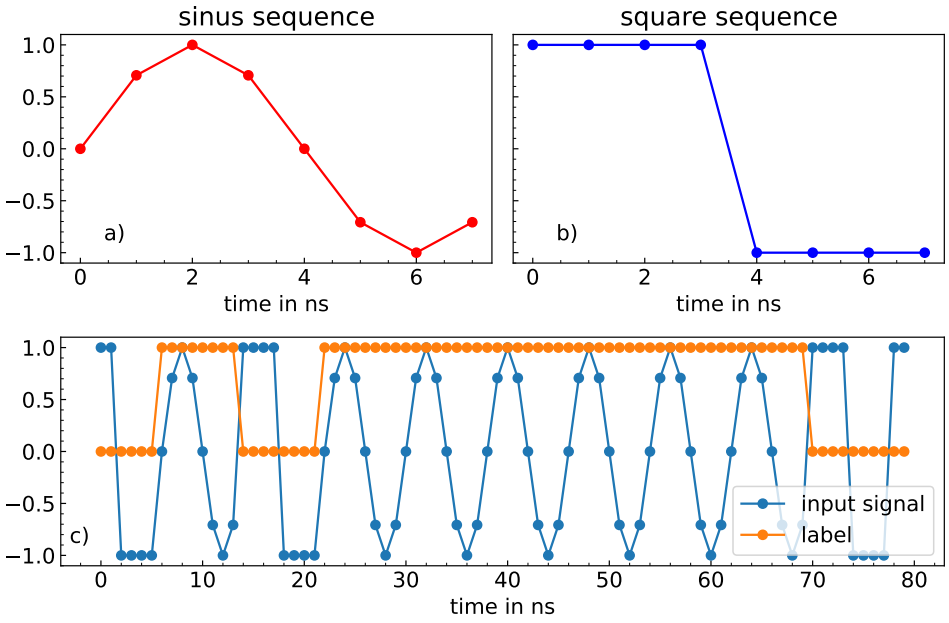


Figure 5.2 – a) Base sequence of sine wave versus time. b) Base sequence of square wave points versus time. c) Example of a sequence of 80 input points versus time in blue and the associated label sequence in orange

$$x(t+\delta t) = x(t) + \delta t \frac{dx}{dt} \tag{5.4}$$

5.5 . Task : discriminate between sine and square

In order to test the behavior and capacity to train a small network of dynamical spintronic neurons, I chose a task that, while simple to solve, is designed to probe the memory and non-linearity of recurrent networks. This task aims to discriminate in a time series which points belong to a square wave and which belong to a sinusoidal wave of the same frequency and amplitude. This task is a benchmark for small networks and reservoir computing [113], [122]. If a point comes from a square signal, it is associated with label 0, and if it comes from a sinusoidal signal, it is associated with label 1. Here, I chose for the full training set to be a single sequence of 80 bits of sine or square waves, with each bit being made of 8 points. Each sequence of 8 points corresponds to a period of the sine or square wave, see Figure 5.2 a) and b). In total we obtain a set of 640 points to predict. The testing set is created in a similar way with a different random seed.

As we can notice in figure 5.2 in the two signals sine and square there are points $+1$ and -1 in position $t = 2ns$ and $t = 6ns$, thus to assign the correct label to these points the network must be able to discriminate between identical points based on the previous point values. In consequence this task is a good test to evaluate if a network presents memory properties. Moreover this task cannot be solved with 100% accuracy with a linear classifier, it's then a good test to evaluate the presence of sufficient non-linearity of a network.

5.6 . Network architecture

In this chapter, we consider a monolayer network for simplicity; a multilayer network architecture will be proposed in the next chapter. To try to solve this task with spintronic dynamical neurons, I chose to start with the simplest possible architecture. The network is made of a layer of a small number of spintronic neurons, between 1 to 48, all obeying Equation 5.5. Weights and biases are non-dynamical quantities here; they represent perfect static devices that don't interplay with the time-dependent nature of the input. This choice has been made to simplify the study of small spintronic neuron networks, but we could also have dynamical behavior for synapses with physical devices.

These neurons receive the external input signal $I_0(t)$ weighted by a matrix W_{ext} represented by the purple box in Figure 5.3; this signal is also amplified with a factor S_{ext}. The outputs of the neuron layer are reinjected into the neurons and weighted by a matrix W_{int}, creating intra-layer couplings of the neurons. The diagonal terms of this matrix are set to 0 to remove autocoupling and thus prevent exponential decay or increase of the neurons. This matrix is represented by the green box in Figure 5.3. Similarly to S_{ext}, a factor S_{int} is applied on the output of W_{int} to tune the strength of coupling. These two parameters S are kept fixed during training. Neurons also receive biases to ensure that they are above the activation threshold when an input is sent. The output of the layer of neurons is then transformed into a one-dimensional output by the matrix W_{out}, represented by the pink box. This output is fed to a binary cross-entropy loss function after the application of a sigmoid function. This operation transforms the one-dimensional output into a probability of being a label 1 or 0. To help classification, a multiplicative factor equal to 1000 is applied to the output before the sigmoid; this sharpens the sigmoid and thus helps separate more clearly between 0 and 1 predicted labels.

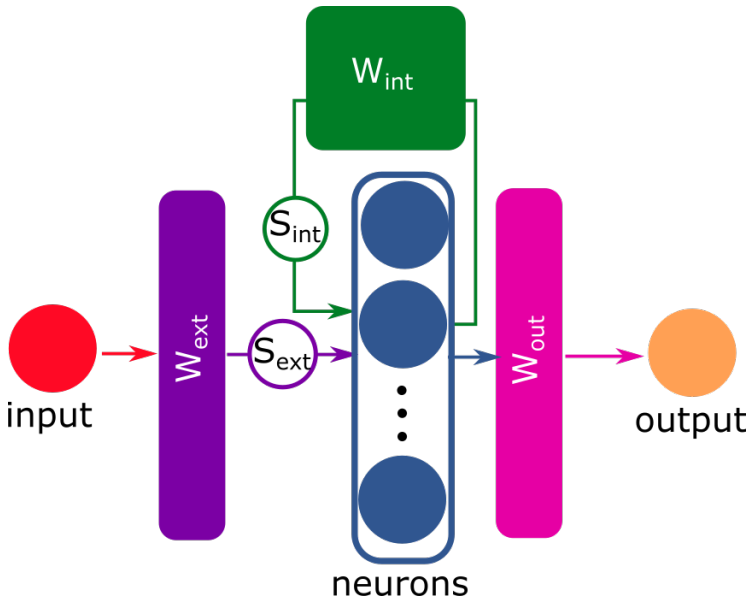


Figure 5.3 – Architecture of a spintronic network : A layer of neurons receives inputs weighted with a matrix W_{ext} and has internal connections weighted with W_{int}. The output of the neurons is transformed into the final output through a final weight matrix W_{out}.

$$\begin{aligned} &\frac{dx_i}{dt} = -\gamma x_i + I(t)x_i(1-x_i) \\ &\text{with } I(t) = S_{ext} \times (W_{ext})_i I_0(t) + S_{int} \times (W_{int})_{ij} x_j + b_i + b_{fixed} \\ &Loss = BCE(Sigmoid(1000 \times (W_{out})_i x_i + b_{out})) \end{aligned} \tag{5.5}$$

5.7 . Backpropagation through time

5.7.1 . Backpropagation through time algorithm

In order to train the dynamics of the network the error needs to take into account the network response through time to a time-series. The algorithm of back propagation through time [92] is made for such situation. This method consists in unfolding the network computation through time to compute the gradient descent versus trainable parameters. Let's illustrate this with a simple example of dynamical network where the state of the network is represented by $x(t)$ obeying Equation 5.6, where W_{int} and W_{ext} are tunable weights applied respectively on the internal step at previous timestep $x(t-1)$ and on an external input $I(t)$.

$$\begin{aligned}
&\frac{dx(t)}{dt} = (W_{int} - 1) \times x(t) + W_{ext} \times I(t) \\
&x(t) = x(t-1) + ((W_{int} - 1) \times x(t-1) + W_{ext} \times I(t))\delta t \\
&x(t) = (W_{int} \times x(t-1) + W_{ext} \times I(t))\delta t
\end{aligned} \tag{5.6}$$

We want to perform back propagation through time after two time-steps. To modify the weight values W_{int} and W_{ext} we need to compute the derivative of the loss function L versus these coefficients. Indeed the gradient descent general structure is to adjust a trainable parameter such as W_{ext} by subtracting it the term $lr \times \frac{\partial L}{\partial W_{ext}}$, where lr is the learning rate. The derivative $\frac{\partial L}{\partial W_{ext}}$ can be developed with the chain rule as Equation 5.7. Similar decomposition is performed for W_{int}.

$$\frac{\partial L}{\partial W_{ext}} = \frac{\partial L}{\partial x(t = 2ns)} \frac{\partial x(t = 2ns)}{\partial W_{ext}} \tag{5.7}$$

All backward paths from the final loss L, represented in pink in Figure 5.4 b), needs to be taken into account to compute $\frac{\partial L}{\partial W_{ext}}$. These multiple paths are visible in $x(t = 2ns)$, indeed it depends on W_{ext} both directly and through the previous time step $x(t = 1ns)$, see Equation 5.8 where the time dimension is completely unfolded. We will split $x(t = 2ns)$ in two terms to represent the two dependency, the direct dependency will be noted $x_a(t = 2ns) = W_{ext} \times I(t = 2ns)\delta t$ and the indirect one $x_b(t = 2ns) = W_{int} \times x(t = 1ns)\delta t$. The derivative of the loss now becomes 5.9

$$\begin{aligned}
&x(t = 2ns) = (W_{int} \times x(t = 1ns) + W_{ext} \times I(t = 2ns))\delta t \\
&x(t = 2ns) = (W_{int} \times (W_{int} \times x(t = 0ns) + W_{ext} \times I(t = 1ns))\delta t + W_{ext} \times I(t = 2ns))\delta t
\end{aligned} \tag{5.8}$$

$$\begin{aligned}
\frac{\partial L}{\partial W_{ext}} &= \frac{\partial L}{\partial x(t = 2ns)} \left(\frac{\partial x_a(t = 2ns)}{\partial W_{ext}} + \frac{\partial x_b(t = 2ns)}{\partial x(t = 1ns)} \frac{\partial x(t = 1ns)}{\partial W_{ext}}\right) \\
\frac{\partial L}{\partial W_{ext}} &= \frac{\partial L}{\partial x(t = 2ns)} (I(t = 2ns)\delta t + W_{int} \times I(t = 1ns))(\delta t)^2)
\end{aligned} \tag{5.9}$$

The expression of $x(t = Nns)$ can be computed in the same way obtaining Equation 5.10, where the sum is accounting for all possible paths for the backward pass.

$$\frac{\partial L}{\partial W_{ext}} = \frac{\partial L}{\partial x(t = Nns)} \sum_{i=0}^{n} W_{int}^{n-i-1} \times I(t = ins) \times \delta t^{(n-i)} \tag{5.10}$$

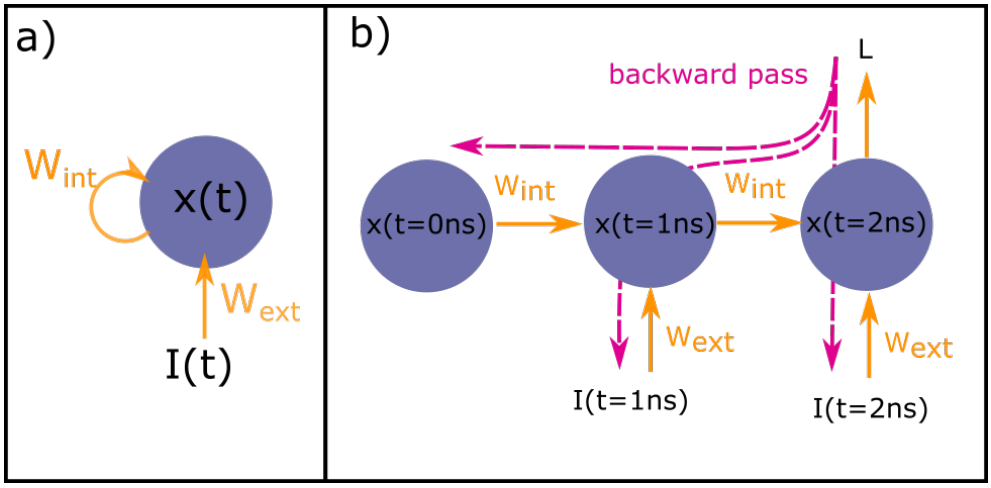


Figure 5.4 – a) Folded representation of a recurrent network, this network receives a time series of inputs $I(t)$ weighted by W_{ext}, a recurrency loop is performed through the weights W_{int}. b) Unfolded representation of the same network, in pink the backward pass paths are represented.

5.7.2 . Pytorch implementation

PyTorch [123] is a Python package designed to implement a machine learning framework in Python. It provides tensorial operations written in C that permit fast computation for neural network training. PyTorch also includes standard functions and classes for machine learning, such as activation functions and parameterized layers of weights. Finally, PyTorch is compatible with graphic processing units (GPUs), enabling the implementation of parallel computations of large arrays or tensors to further speed up neural network training.

PyTorch has a framework named autograd that implements automatic computation of the gradients of the loss with respect to trainable parameters. All trainable parameters of a network are called leaf tensors and are the values from which all gradients will be calculated. During the forward pass, all the operations performed on the leaf tensors and the resulting intermediate values are stored in the "$.grad_{fn}$" attribute of the final outputs. The computational graph describing how the input quantities are transformed into outputs is then recreated from these stored operations and intermediate values. Running through this computational graph from outputs to leaf tensors and applying the derivative chain rule permits the evaluation of the gradients of the loss with respect to all leaf tensors.

5.7.3 . Gradient issues

Table 5.1 – Accuracy obtained on 20 initialisations for different gradient clipping strategies

	fixed clipping	rescaled gradient clipping
Accuracy	$75.18 \pm 6.31\%$	$72.62 \pm 2.74\%$

One of the issues with back propagation through time is the uncontrolled growth of the gradient values when unfolding the time dimension. Indeed as can be observed in equation 5.10 some terms in the gradients are multiplied by themselves for each unfolding step, here $\sum_{i=0}^{n} W_{int}^{n-i-1}$. These terms at a high power when time steps are numerous leads to this so called "exploding gradient" values [124][125]. This leads to first very high value of gradients that leads to too big updates of the parameters, then if nothing is done the software simulation can fail if values exceed the available digit range. To solve this issue gradients in the network needs to have their gradients clipped. The most standard method is to clip the norm of the gradients, meaning that if the norm of the gradients exceed a certain threshold the gradients are rescaled down to set the norm to the threshold value. However a simpler clipping strategy worked slightly better in my case, if any components of the gradient exceed the threshold it's clipped to the threshold value. In the case of multilayer network, developed in the next chapter, the fixed clipping value gives an accuracy of $75.18 \pm 6.31\%$ and with a rescaling of the norm of the gradient the network only reaches $72.62 \pm 2.74\%$.

In conclusion gradients need to be clipped to a limit value to prevent divergence of their values during the time unfolding of the backward pass.

5.7.4 . Testing different back-propagation scheme

The chosen optimizer in all the following sections is the ADAM optimizer [126]. This algorithm implements a stochastic gradient descent taking into account first and second order momenta of the gradients.

IIn this section, each training method is applied to a spintronic network of 24 neurons with 20 different seeds of initialization, deactivating the recurrent loop W_{int} for simplicity, with $batch = 16$. After ensuring correct values for γ, S_{ext}, and b_{fixed} to activate all neurons and optimizing the learning rate, we test different methods with the same number of weight updates, here 90 updates.

The standard backpropagation through time scheme consists of presenting the full input signal to the network and backpropagating only at the last

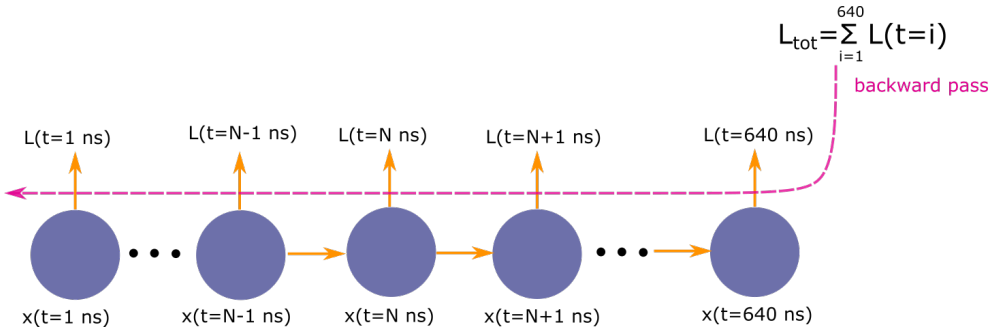


Figure 5.5 – Unfolded representation of the backpropagation through time standard scheme. Here the loss is computed at each time step and summed, backpropagation is performed on the full signal.

timestep. In our case, we would sequentially present all the 640 input points and record the error on the predicted label for each point. The final error is the sum of all 640 computed errors. After calculating the total loss, we backpropagate this loss on the network's parameters, unfolding through the 640 previous points in time, as we can see in Figure 5.5. After running 90 epochs, we obtain a mean accuracy of 87.85% on the test signal. However, some networks reach 100% accuracy, so it's not an issue of network computing capacity, but mostly of failing training. This low value is due to an issue linked to the exploding gradient problem. While we addressed the exploding gradient problem, high powers are still present in the gradients, often leading to chaotic behavior of the gradient computation if many timesteps are computed. From another point of view, this means that the loss function presents a landscape with many local minima and is difficult to optimize via gradient descent. A possible solution is to truncate the backpropagation after a fixed number of timesteps to limit the higher power terms.

This scheme called truncated backpropagation through time is another standard technique [127] where we define two number $k1$ and $k2$. The signal is sent to the network sequentially and every $k1$ points the accumulated loss on the last $k2$ points is backpropagated. Generally we chose $k1 = k2$. Here in our case we chose $k1 = k2 = 30$, meaning that while sending the input signal, every 30 input points we backpropagate the loss accumulated on these last 30 points, see Figure 5.6. After each backpropagation step the loss is set back to 0. To prevent non uniformity coming from performing the backpropagation always at the same points $t = n \times 30$, an additional shift of 1 is added at each epoch, thus performing backpropagation at $t = n \times 30 + i_{epoch}$. Here we tested this method in the same configuration as before except for the learning rate that is modified to a more adapted value and the number of epoch set to 3 to keep a similar number of backpropagation performed, thus a similar number

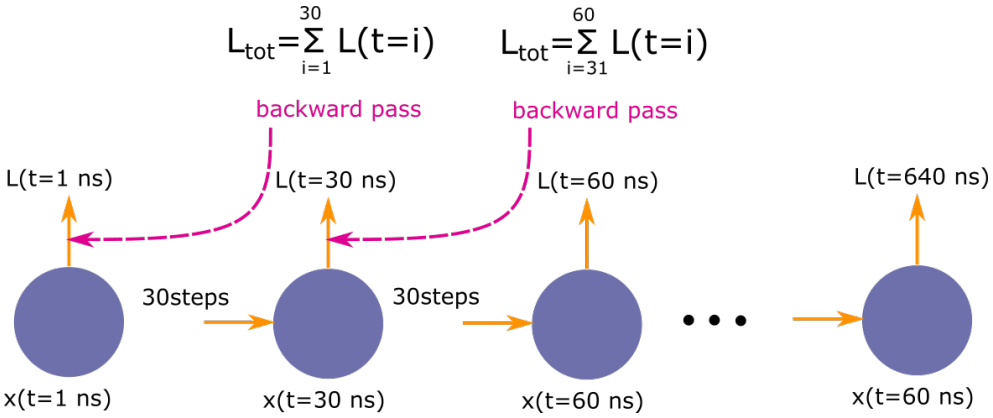


Figure 5.6 – Unfolded representation of the truncated backpropagation through time standard scheme. Here the loss is computed at each time step and summed over 30 time steps. Backpropagation is performed at the end of each 30 points window only on this last 30 time steps.

of weights updates. With this method we improved the accuracy to 98.15%.

In this truncated backpropagation a small issue is degrading the performance of the training. When the backpropagation if performed at timestep $t = 30ns$ the network parameters are modified. However the internal state of the network comes from the time evolution of the starting point $x(t = 1ns)$ with the previous parameters over 30 points. This starting point is different to what it would be in the testing phase where the evolution is performed from $x(t = 1ns)$ with the final parameters. To have a more correct estimation of the error on points 30 to 60 we would need to start from $x(t = 1ns)$ evolved for 30 points with the new parameters. However recalculating the correct starting points for each backpropagation section would require a lot of computation and time in software as we need for each starting point $x(t = N \times 30ns)$ to calculate the previous $N \times 30$ previous points. It would be easier in hardware as the point $x(t = N \times 30ns)$ could be produced by just injecting the $N \times 30$ first inputs which takes only a small amount of time.

I then propose an improvement of the truncated BPTT described in figure 5.7. The goal is to minimize the error on the starting point of each 30 point sequence of TBPTT. To do this I start as in standard TBPTT from the starting point $x_0(t = 1ns)$ where the subscript 0 correspond here to the state of the network's parameters, here 0 update through backward pass. This point is then evolved over 30 points with the inputs from time 1ns to 30ns. Backpropagation of the total loss is then performed. The state of the network's parameters is now 1. The next window of 30 points covers inputs from t=2ns to t=31ns, it thus needs to start from $x_1(t = 2ns) = x_0(t = 1ns) + f_1(x_0(t = 1ns))\delta t$ built

Table 5.2 – Accuracy obtained on 20 initialisations for different backpropagation schemes

	standard BPTT	truncated BPTT	smooth TBPTT
Accuracy	$87.85\% \pm 10.54$	$98.15\% \pm 5.56$	$100\% \pm 0$

by evolving $x_0(t = 1ns)$ with the new network parameters. This starting point now evolve through the 30 inputs from t=2ns to t=31ns and backpropagation is performed. In the same way the next starting point will be generated from evolving $x_1(t = 2ns)$ with the new obtained network 's parameters state 2 $x_2(t = 3ns) = x_1(t = 2ns) + f_2(x_1(t = 2ns))\delta t$. Here we accumulate a small error as $x_1(t = 2ns) \neq x_2(t = 2ns)$, however this error is smaller than if we had accumulated it on a full window of 30 points as in the previously standard TBPTT. The general rule to generate the new starting point of a sequence of TBPTT from the previous starting point is Equation 5.11. This procedure by minimizing discontinuity in the starting points of the TBPTT windows permits to reach an accuracy of 100% for all the 20 network initialisations. This method is designed for time series where the labels or targets aren't independent and also form a coherent time series. This is mostly a software trick, in hardware the previous implementation of BPTT where the starting point is physically produced by the system is simpler to implement. It's this final backpropagation scheme that I employed in the following section.

$$x_{n+1}(t = start(ns)) = x_n(t = start - 1(ns)) + f_{n+1}(x_n(t = start - 1(ns)))\delta t \tag{5.11}$$

To summarize backpropagation through time can be quite challenging to use. Due to the time recurrence gradients must be clipped to prevent divergences. Moreover the backpropagation scheme needs to be adapted to the nature of inputs and outputs. I thus proposed an ameliorated version of truncated backpropagation through time where the starting point of each BPTT window is calculated from the previous starting point to minimize accumulation of error between parameters updates. This leads to better performances where a network of 24 spintronic neurons can solve the sin square task in 100% of cases.

5.8 . Different types of neurons

To evaluate the computational power of the spintronic neuron network, I compared it to networks composed of dynamical neurons with other diffe-

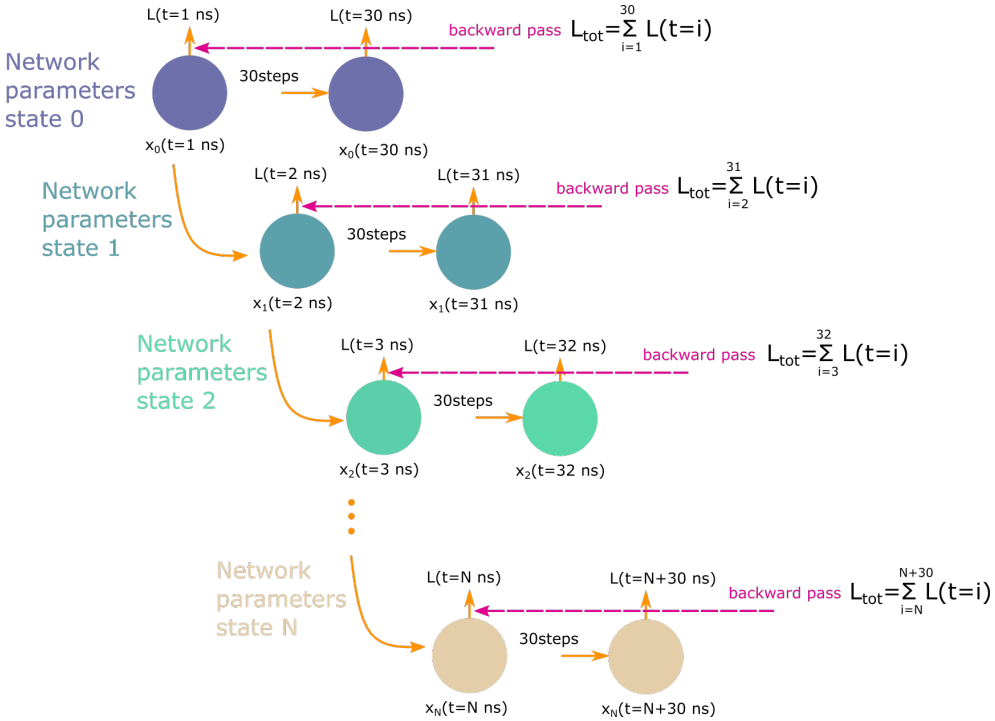


Figure 5.7 – Unfolded representation of the adapted truncated backpropagation through time standard scheme. Here the loss is computed at each time step and summed over 30 time steps. Backpropagation is performed at the end of each 30 points window only on this last 30 time steps. the notable difference is that the starting point of the 30 points window is computed from the previous starting point after one time step.

rential equations. These ODEs are displayed in Equation 5.12. The first one is the simplest damped dynamical neurons with a drive that doesn't depend on the neuron internal value x. The second one has a drive linear in x, the third one is non-linear in x, this is comparable to the spintronic neuron in order of x. Comparing these neurons provides insights of which terms in the drive lead to good performances.

$$\begin{aligned}
&\frac{dx}{dt} = -\gamma x + I(t) \text{ simplest drive with relaxation} \\
&\frac{dx}{dt} = -\gamma x + I(t)x \text{ simplest liquid-time drive with relaxation} \\
&\frac{dx}{dt} = -\gamma x + I(t)x^2 \text{ simplest non-linear liquid-time drive with relaxation} \\
&\frac{dx}{dt} = -\gamma x + I(t)x(1-x) \text{ spintronic neuron}
\end{aligned} \tag{5.12}$$

As we can see in figure 5.8 a) the neurons of all types respond non-linearly to a sinusoidal drive. When submitted to a constant drive we see that the non-spintronic neurons present a diverging trajectory, however to keep a more physical behavior neurons responses need to be clipped to a fixed range, I chose it to be $[0, 1]$ for simplicity. If a neuron is reaching a value outside this range it has to be clipped. Its gradient should also be clipped to 0 if this neuron stays in saturated states $x = 0$ or $x = 1$ as the main term in all gradients $\frac{dx(t=n)}{dx(t=n-1)}$ is now 0. Testing this procedure on unbounded ODEs such as presented in Equation 5.12, showed that networks with such neurons are not efficiently trainable, as displayed in Figure 5.8 c) the final loss is much lower for spintronic neurons compared to the three other neuron types. This can be explained by the fact that these neurons rapidly reach their clipping values. At this point gradients of the loss are clipped to 0 leading to no update of the coefficients connecting these neurons. On the contrary the spintronic neuron equation is naturally bounded between $[0, 1]$ as the input drive becomes continuously 0 when the neuron approach limit values 0 or 1. This results in gradients to be always correct as the neurons output values doesn't require clipping. The spintronic neuron equation is one of the simplest ODE that can be built to produce an output naturally bounded between $[0, 1]$, the fact that the bounding arising from the ODE appears to be the main reason why this network can be trained despite the constraints on x. The order in x terms doesn't seem to be a critical parameter, however an order at least 2 is required to present a naturally bounded dynamics.

It is important to note that the three non-spintronic networks could be trained efficiently when removing the clipping constraint ; however, this would not be coherent with a physical behavior. In conclusion, neurons with natu-

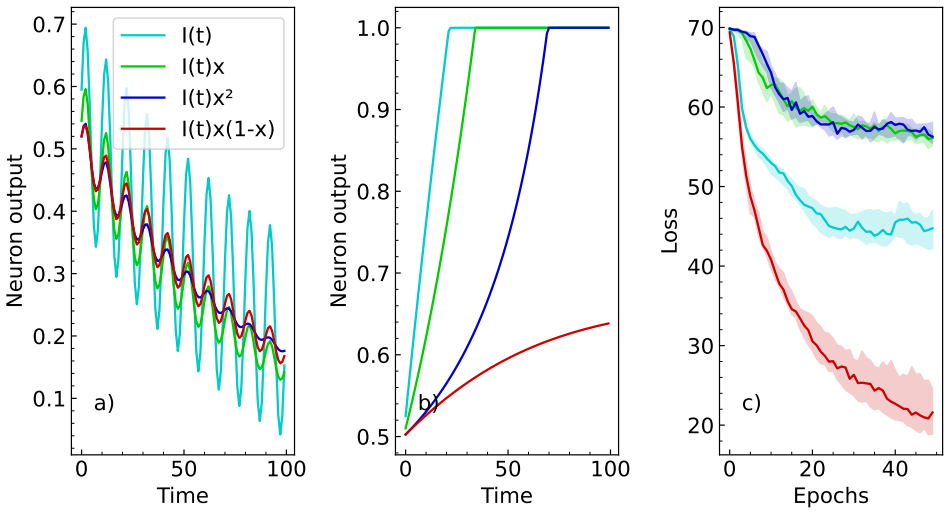


Figure 5.8 – a) Neuron output for different types of neurons when submitted to a sinusoidal drive. b) Neuron output for different types of neurons when submitted to a constant drive. c) Loss during training on the sine square task for the four different types of neurons.

rally bounded dynamics are trainable, while forced clipping of unbounded dynamics prevents efficient training. Networks made of physical neurons can be trained despite their bounded behavior, as this behavior naturally arises from the dynamics of the physical devices.

5.9 . Comparison between reservoir computing and BPTT trained network

In order to evaluate the efficiency of training our network of spintronic neurons, we compare this approach to a reservoir computing scheme applied to the same network. Training with our backpropagation scheme is more efficient, specifically for a low number of neurons in the network. In the case of reservoir computing, the network is initialized with random values for the first layer of weights going from the input signal to the neurons; then, the full input signal is fed to the network, and outputs are recorded. Finally, output weights are computed via the pseudo-inverse operation. It is important to note that this operation permits finding the best linear mapping between neurons' outputs and the predicted label, so there is no sigmoid function applied on the final output in the case of reservoir computing. In the case of a trained network, the procedure is the same as in the previous section; for both configurations, different numbers of neurons in the network are tested. Results are displayed in Figure 5.9. For both reservoir computing and the trained network,

two types of networks have been tested : one without intra-layer connections so $W_{int} = 0$ and one with intra-layer connections. Looking at the mean accuracy over different initializations, Figure 5.9 a), a clear trend appears : back-propagation through time performs better than reservoir computing for the same number of neurons. Moreover, adding intra-layer connections improves performance in the case of reservoir computing; this is expected as it adds more non-linearity to the network. There is no notable difference in BPTT, which is understandable as the simplest network with no intra-layer connection is already complex enough to solve the task most of the time.

A notable feature for both reservoir computing and trained network is that with the same number of neurons there can be a high dispersion of final accuracy, see figure 5.9 b). For instance with the reservoir of 48 neurons without intra-layer connections nearly all networks initialisations have an accuracy above 99.5% but one of them as an accuracy of 53.75%. This dispersion is more important for smaller networks and for reservoir computing. This points a responsibility in the initialisation values of the weights, indeed the smaller the network is the less diverse are the weight values in a network at initialisation, and thus the network is likely to be less flexible to trained to a task. In these cases BPTT is performing better as the first layer of weights is also tuned and can compensate some of the bad initialisations, while reservoir computing keeps this first layer in initialisation state. If we have a closer look at the diversity of results the difference of performance is even greater between reservoir computing and BPTT. In figure 5.9 b) the median accuracy is represented in solid lines and the extend between maximum and minimum accuracy for a same number of neurons in light colored area. We can thus observe that the two types of BPTT trained networks can reach 100% accuracy with only two neurons and that the median accuracy reaches 100% for four neurons. In the case of the reservoir computing, with intra-layer coupling we need 16 neurons to reach 100% in best configuration, without intra-layer coupling we need 24 neurons, and in both cases even with 48 neurons we cannot reach a median of 100%.

Taking the example of a network of two neurons solving perfectly the sin square task we can really understand how the dynamics of neurons augment the computing power of a network. In Figure 5.10 b) the responses of two neurons from such network are represented after application of the weights from W_{out}. These two neuron outputs have quite close behavior when submitted to an input signal. However when we look at the difference of these two signals in Figure 5.10b) and c) we see that the neuron 2 in orange is superior to the neuron 1 in blue when the input is sinusoidal, and inferior when the input is a square wave. This is because the neuron 1 is in fact submitted to a strong

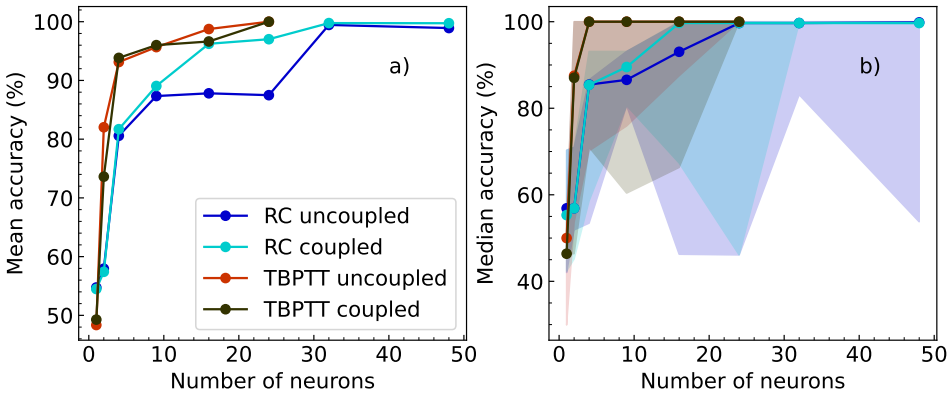


Figure 5.9 – a) Mean accuracy versus number of neurons in the network. Four types of networks are represented, two networks are trained with reservoir computing, one with internal coupling of the neurons and the other without. The two other networks are trained with BPTT, similarly with internal coupling of the neurons and the other without b) Median accuracy in the same configurations.

drive and then its output is scaled down by the W_{out} matrix and the neuron 2 is submitted to a small drive and its output scaled up by the W_{out} matrix. This is an example of how the non-linearity of the dynamics of two spintronic neurons can discriminate between different non-linearly separable inputs when tuned with BPTT.

From this study we notice that training with backpropagation is more efficient than reservoir computing, moreover it is possible to solve the sin square task with down to two neurons with BPTT trained networks.

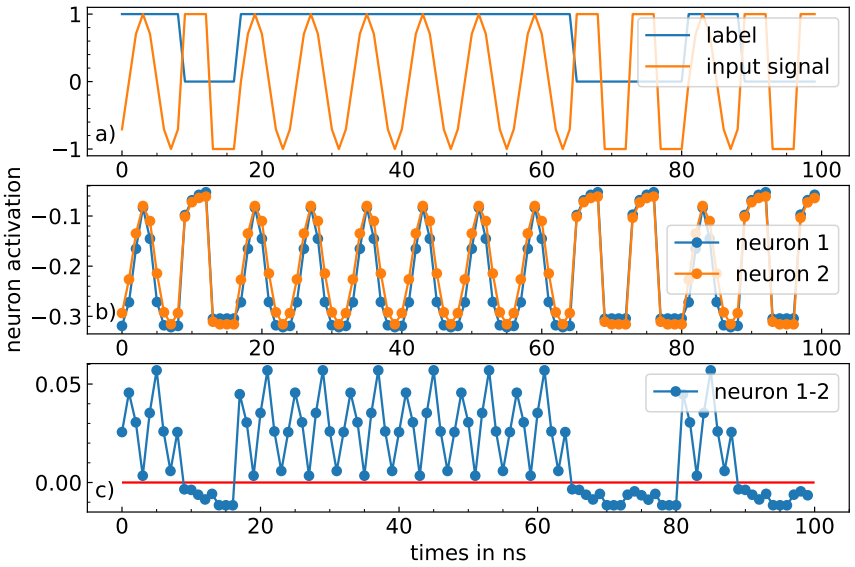


Figure 5.10 – a) Sequence of 100 input points and their corresponding labels b) Weighted response of the two neurons of the network by the weights of the final layer W_{out}. c) Difference between the two neurons responses in blue and threshold for label discrimination in red.

5.10 . Conclusion

In this chapter, I demonstrated that a network of spintronic neurons can solve a non-linear and temporal task requiring memory, to discriminate between sine and square waves. This network is trained with an updated version of truncated backpropagation through time that I designed to yield better performance on time series prediction. Finally, training device drives with this method is more efficient than considering the network as a reservoir. Especially with a small number of devices, using backpropagation, we can obtain 100% accuracy with only two neurons.

6 - Multilayer network

6.1 . Summary

The goal of this chapter is to demonstrate a multilayer architecture for STNOs dynamical network on a complex task. The other objective is to investigate what are the constraints to train such network, especially in terms of matching between input timescale and device characteristic response time. The proposed network architecture is based on the previously described single layer network of coupled STNOs. However inputs to each layer need to be centered to limit the saturation of the STNO's outputs, to do so high-pass filters are applied on the outputs of each STNOs. The tackled task is sequential DIGIT : handwritten 8x8 digit images [118], are send pixel by pixel to the network and the correct label must be predicted after seeing the full series. After hyper-optimisation of the learning rate and relaxation time of devices at fixed input timescale, the network can reach $89.83 \pm 2.91\%$ classification accuracy. This performance is equivalent to the $89.00 \pm 3.48\%$ accuracy reached by a standard CTRNN with same number of neurons and the *tanh* activation function. We also derived two rules to ensure good training of this network. First, the relaxation time of devices must be larger than the input timescale to ensure enough memory in the system. Second, the drive duration product, involving the network's weights and the input timescale, must be close to one. This second rule limits the saturation of the neurons by the upper or lower bounds. It is important to note that parameters such as the learning rate or the STNOs relaxation time can be varied in a five fold range around their optimal value without accuracy degradation. The network is capable to adapt to a variable range of input dynamics. Another important results is that the density of connections in the network can be lowered down to 50% without accuracy degradation, which could be an advantage for future hardware implementations.

6.2 . Introduction

In this chapter, we demonstrate the chaining of successive layers of spintronic dynamical neurons. The output from a layer of spintronic neurons is re-injected as input into the following layer of neurons. In this configuration, as the signal is transmitted through multiple layers, it undergoes successive non-linear and time-dependent transformations driven by the neurons' ODE. Coupling two similar ODEs gives rise to dynamics that cannot be achieved with a single equation. Thus, we expect that chaining neurons will increase com-

putational power compared to a single-layer network. To test this increase in computational power, we choose a task which requires both non-linearity and memory and which is harder than Sine-Square : recognizing handwritten digits from the Sequential DIGIT dataset [118].

This is the first demonstration of a multilayer network using STNOs as dynamical neurons. Moreover, the architecture of this network is compatible with hardware implementation and considers the constraints of physical neurons, such as their internal timescale and bounded dynamics.

I start by presenting the network architecture and the task. Then, I describe the training procedure and hyperparameter optimization, before presenting the first training results. I then show the results of a more in-depth investigation of critical parameters, such as the density of connections and the correlations between the neurons' characteristic relaxation times, the input timescale, and the learning rate, to obtain maximal accuracy.

6.3 . Network Architecture

We propose an architecture based on the single-layer network presented in the previous chapter with some adaptations to chain layers. The resulting network is represented in Figure 6.1, where each neuron i in layer $n+1$ obeys Equation 6.1. Neurons, represented by blue circles in the architecture, receive several different driving terms summed as a single input I, and produce an output labeled x_i^{n+1} for neuron number i in layer number $n+1$. This output is then processed by a high-pass filter represented by an orange rectangle in Figure 6.1. The filter's role will be discussed in the following section in more detail, but its main purpose is to produce an output y_i^{n+1} for the neuron x_i^{n+1} suitable to be re-injected into another layer.

$$\begin{aligned} &\frac{dx_i^{n+1}}{dt} = -\gamma x_i^{n+1} + I(t)x_i^{n+1}(1-x_i^{n+1}) \\ &\text{with } I(t) = S \times ((W_{ext}^n)_{ij} y_j^n + (W_{int}^{n+1})_{ij} y_j^{n+1} + b_i^{n+1}) + b_{fixed} \end{aligned} \tag{6.1}$$

Let's now have a look at the different terms in the input received by a neuron. Layers are connected to each other by a weight matrix W_{ext}^n that unidirectionally couples neurons from layer n to layer $n+1$. These forward connections are represented by purple boxes in the architecture figure. Thus the neuron x_i^{n+1} receives weighted filtered outputs from layer n represented by the term $(W_{ext}^n)_{ij} y_j^n$. Consequently, matrix W_{ext}^n has the size of the receiving layer in rows and the size of the input layer in columns. Here, this corresponds to a 32×32 matrix. Within each layer, neurons are coupled by a weight

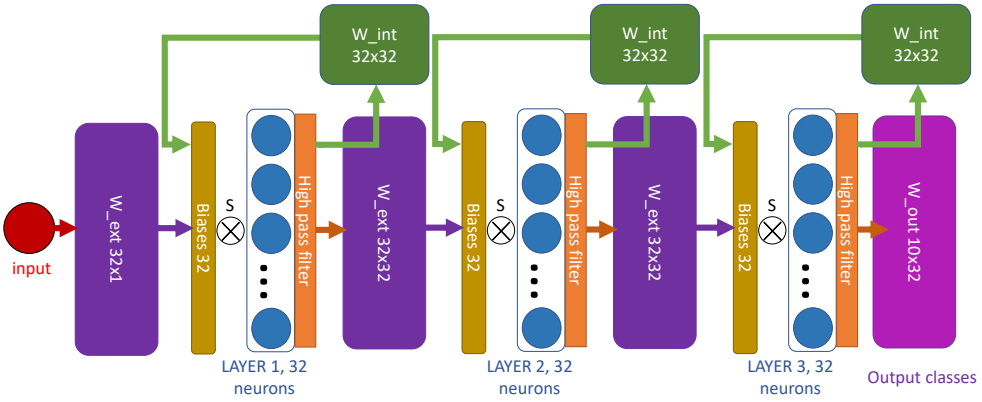


Figure 6.1 – Architecture of the network made of 3 layers of 32 dynamical neurons. Each layer receives inputs from itself via the green recurrent loop W_{int}, and from the previous layer via the purple weight matrix W_{ext}. Outputs from a layer are filtered with a high-pass filter (orange box).

matrix W^n_{int}, represented by the green box and arrows. In consequence the neuron x_i^{n+1} receives weighted filtered outputs x_j^{n+1} from layer n represented by the term $(W^n_{ext})_{ij}y^n_j$. Similarly this matrix has a size 32×32 since it couples 32 oscillators to each other. Self-coupling is removed to prevent exponential increase or decay of an oscillator independent of the task input. In consequence, we set all diagonal weights of the W_{int} matrices to zero, thus having $32 \times 32 - 32 = 32 \times 31$ tunable weights. The remaining terms in the input are additional sets of 32 tunable biases b_i^{n+1} and a fixed bias b_{fixed} applied on each layer $n+1$ of neurons. These biases are represented by the dark yellow boxes in figure 6.1. These biases are equivalent to setting a threshold in the activation of neurons. Finally, we apply a scaling factor S to the input received by each neuron, this factor has a similar role to the high-pass filter detailed in the following section. It's important to note that I chose this factor as an hyperparameter and it is fixed during training.

The first and last layers slightly differ from the general geometry as they respectively receive the input from the dataset and produce the output classes. The input is a time-series with a single value at each time step. The first layer receives the weighted single input value through W^1_{ext}, thus this matrix has size 32×1. Similarly the output of the final to a layer of 10×32 weights and 10 biases, going to the 10 output classes. A LogSoftMax function is then applied to compute the logits without using additional neurons as non-linearity for

the ten final outputs.

6.3.1 . Role of the high-pass filter and amplification factor

What appears when chaining dynamical neurons is that the output from a neuron is not of the same amplitude as the input it received. To illustrate this issue, we chain 3 neurons. I inject a sinusoidal signal of normalized amplitude 1 and period of ten nanoseconds in the first neuron, which has a relaxation frequency of $\gamma = 0.15GHz$, see Figure 6.2 a). The output of this neuron is then fed to another similar neuron to simulate a second layer. Similarly the output of this second neuron is sent to a third neuron to estimate the change in signal in layer 3. The behavior of each neuron is reported in Figure 6.2 c). A first observation is that the first neuron has a an oscillation amplitude around 4 times smaller than the input signal. This is due to the term $Ix(1-x)$ in Equation 5.3. Indeed the value of $x(1-x)$ is roughly 0.25 when the neuron has a mean value 0.5, thus dividing the signal amplitude by approximately 4. In the same way the amplitude of oscillation is once more decreased in the second neuron and thus quasi-nonexistent in the last neuron. This is an issue since the time-varying nature of the input signal is quickly reduced by chaining layers then discarding most of useful information.

We tackle the amplitude loss through layers by amplifying the output of a neuron by scaling it with a coefficient that we call S. Let's go back to our previous example of three chained neurons and choose $S = 4$ to compensate for the amplitude loss; this study is represented in Figure 6.2 d). In this configuration, the amplitude of oscillation of the second neuron is now much higher and quite similar to that of the first neuron. However, the third neuron still shows a loss of amplitude. This time it is because the neuron starts to approach saturation $x = 1$; in this case, the drive multiplied by $x(1-x)$ goes to a value close to 0. The neuron has been pushed toward 1 because the mean value of the neuron output has also been scaled by $S = 4$, adding a positive offset drive. We thus want to discard the offset coming from the mean value of the previous neuron.

The easiest solution to discard an offset from a time-varying signal is to pass this signal through a high-pass filter. This solution can be implemented in hardware with simple electronics, making it realistic to use with physical devices. This kind of filter is designed to let high frequencies pass while blocking low ones. Applying the high-pass filter in simulation requires the addition of another dynamical variable y_i^n, which corresponds to the filtered output of neuron x_i^n. This filtered output evolves according to Equation 6.2. In this equation, y_i^n is the filtered output of neuron number i of layer n x_i^n, and f_{cut} is the

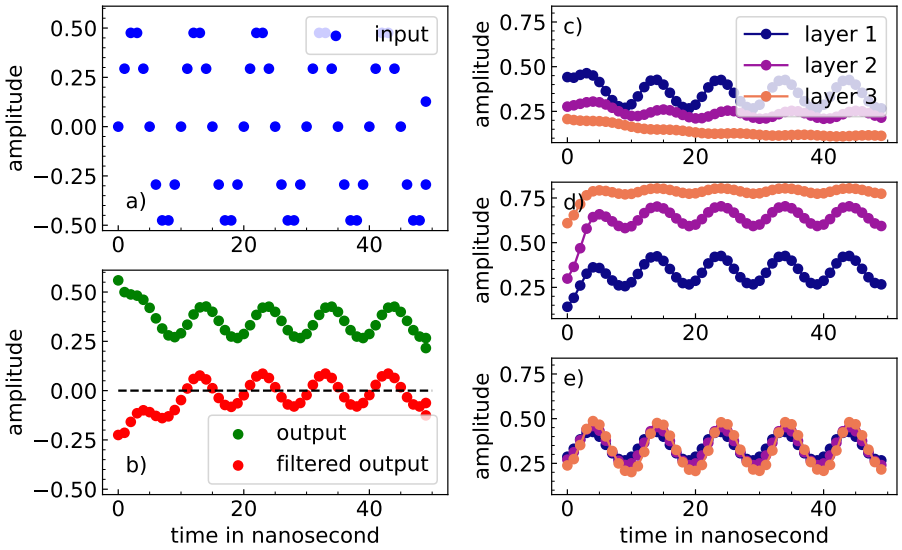


Figure 6.2 – a) Amplitude versus time (nanoseconds) of a sinusoidal input signal of period $T = 10ns$
b) Response of a neuron with $\gamma = 0.15GHz$ to the sinusoidal input signal versus time, with or without application of a high-pass filter c) Response of three chained neuron with $\gamma = 0.15GHz$ to the sinusoidal input signal versus time d) Response of three chained neuron with $\gamma = 0.15GHz$ to the sinusoidal input signal versus time, with 3 times amplification between layers e) Response of three chained neuron with $\gamma = 0.15GHz$ to the sinusoidal input signal versus time, with application of a high-pass filter and 3 times amplification between layers

cutoff frequency below which the filter doesn't transmit the signal. Applying a high-pass filter with $f_{cut} = 0.05$, the output signal of a neuron shown in green in Figure 6.2 b) is transformed into the signal in red, where the amplitude of oscillation is conserved and the mean value is shifted to zero. Now, we can amplify the filtered signal without amplifying an offset. In Figure 6.2 e), the three neurons are chained while applying a high-pass filter on the output of each previous layer and with a scaling factor between layers $S = 4$. We now observe that the amplitude of oscillation is conserved through the layers, as all neurons have the same range of values, and the time-varying information of the signal is not lost anymore.

$$\frac{dy_i^n}{dt} = -2\pi f_{cut} \times y_i^n + \frac{dx_i^n}{dt} \tag{6.2}$$

This filtering and amplifying minimizes the number of neurons that are in saturated states, thus ensuring a maximal transfer of useful signal from layer to layer.

This multilayer network presents the same global layer structure compared to the monolayer network : a matrix W_{ext} weighting the inputs, and a matrix W_{int} implementing a recurrent loop inside a layer. However, the chaining of successive layers required some adaptation. The ideal configuration to avoid the saturation of neurons is to send them inputs varying around zero. However, due to their bounded dynamics, neuron outputs have an offset with a value higher than the amplitude of oscillation. To discard this offset and be able to amplify only the time-varying signal, I implemented a numerical high-pass filter between successive layers. This solves the offset issue and is possible to implement in hardware. The other principal difference is due to the chosen task; instead of assigning a label to each time step, here we assign a label to the entire time series of 64 time steps.

6.4 . Task : Sequential DIGIT

The task that I chose is a time-series classification task, sequential DIGIT based on the eponymous DIGIT dataset [118]. The non sequential version of DIGIT is a reference benchmark for small networks. This dataset is made of 1797 8x8 pixels gray scale images representing handwritten digits, see Figure 6.3 a). The task here is to predict the class in which each image belongs when the image is presented sequentially to the network one pixel after one pixel with a time step of 1 ns, see Figure 6.3 b). In this case the label prediction is done only after the final time-step and not at each time step contrary to the sine-square task.

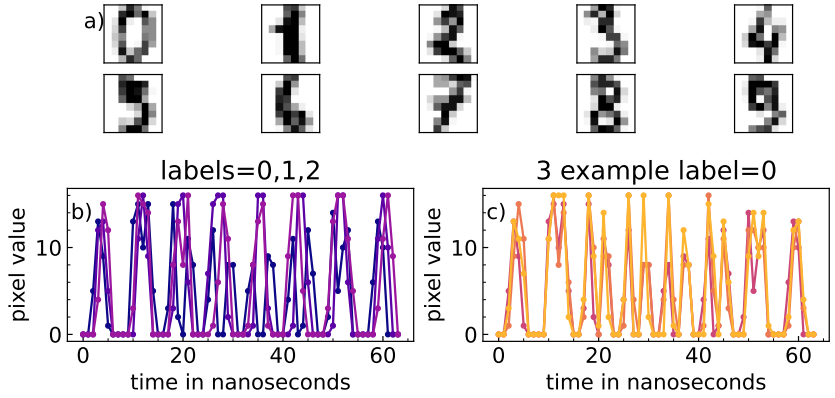


Figure 6.3 – a) Example of each of the ten classes b) Example of images 0, 1 and 2 expressed sequentially in time c) Three examples of images in the class 0

The sequential nature of this computation requires a lot of time in software since the simulated neurons of the network evolves in time and need to be computed for each input. Thus DIGIT was well suited as a small dataset, it helped to keep reasonable the time to perform numerous epochs on several networks in parallel. I split the dataset in two, a training and a testing part in respective proportions 50% and 50%. An epoch is thus constituted of 898 labels and $898 \times 64 = 57472$ points in time .

This task is non-linearly separable as displayed in Figure 6.4, knowing the mean value of the images is not enough to perform classification. This task thus probes the time non-linearity of our system.

6.5 . Training procedure

The training procedure here is slightly different compared to the sine-square task; indeed, we do not want to predict a label for each time step in a time series but to assign a label to the full time series corresponding to one image. For each epoch, the full dataset is run through, with each image being converted into a 64-point time series, see Figure 6.3. This sequence is fed to the network, and after the last time step $t = 64,$ ns, the negative log likelihood loss is applied to the output of the network and the desired target label. The network internal state is reset after each image as there is no time correlation between images; this could be done in hardware by letting the neurons relax in the absence of input. The loss is backpropagated after each batch of 120 images run through the network; this total loss is an average of the 120 individual losses.

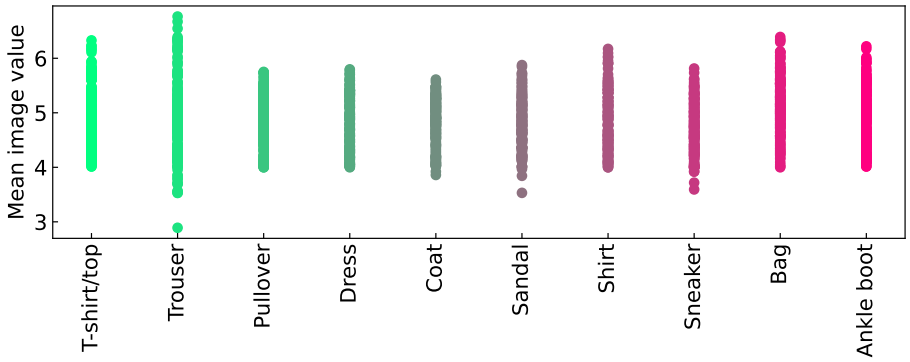


Figure 6.4 – Mean pixel value for all ten image classes, all classes appear non linearly separable

With this configuration, we do not truncate the backpropagation through time as only 64 time steps are performed, but we still need to clip the gradients to prevent exploding gradients. Now that the network is more complex compared to the single-layer case, it is harder for the loss to find a stable minimum; thus, to help the network parameters converge, the learning rate decreases through the epochs as described in Equation 6.3, where n_{epoch} is the index of the epoch and lr_{decay} is the learning rate decay.

$$lr = \frac{lr_0}{\frac{n_{epoch}}{lr_{decay}} + 1} \text{ with } lr_0 = 0.02 \text{ and } lr_{decay} = 5 \tag{6.3}$$

6.6 . Optuna and hyperparameter optimisation framework

Optuna is a hyperparameter optimisation framework based on Bayesian optimisation. The purpose of Optuna is to find the best set of hyperparameters that minimize an objective function. In our case, the objective function is the loss of our network. This objective function is treated as a black box by Optuna, and a probabilistic model is fitted to reproduce the output of the objective function. Specifically, the optimisation algorithm will start by evaluating the performance of some random combinations of hyperparameters. Optuna then fits a model that maximizes the probability density of obtaining the best hyperparameters. This model is based on a sum of Gaussian distributions centered on the best hyperparameter sets. Using this fitted probabilistic model, the algorithm calculates a new set of hyperparameters that should minimize the objective function. Once the true output of the objective function is evaluated with these hyperparameters, the model is fitted once more, inclu-

ding this new configuration if it gives a good result. This process is repeated iteratively until a user-defined number of trials have been tested. An early stop method called pruning can also be activated to stop unpromising trials before the end of the training of a network.

Optuna also provides hyperparameter analysis tools such as maps of the best results versus hyperparameters, the importance of the different hyperparameters, and so on, enabling users to check that optimisation is successful and that it explored meaningful regions.

6.7 . Training results : Obtaining high-performance networks

In this section, I will demonstrate that a network with spintronic neurons can achieve high accuracy on the DIGIT task. We will focus on the optimal number of layers and optimizing the network parameters while preventing saturation of the neurons.

6.7.1 . Impact of the Number of Layers

In recurrent neural networks, contrary to most feedforward architectures, increasing the number of layers does not automatically increase the performance of the network. RNNs generally perform best with 2 or 3 layers. Indeed, for n layers, the n^{th} derivative of the output depends on the input vector; generally, parameterizing up to the 2^{nd} or 3^{rd} derivative is sufficient to get any desired output. I thus investigated how the number of layers in my spintronic neural network impacts the performance.

For each number of layers, I chose to first do a hyperparameter optimization step on three coefficients impacting the neurons' dynamics while keeping a fixed time between two inputs of 1 ns. The optimized coefficients are b_{fixed}, S, and γ, see Equation 6.1. The found coefficients are displayed in Table 6.1. We notice that the amplification factor S does not vary much, but the two other parameters γ and b_{fixed} present a trend. Indeed, the damping γ increases with the number of layers, and the fixed bias added to the input of each layer also increases with the number of layers. These two parameters have opposite effects, one being a positive damping and the other one a negative damping, and they define the equilibrium point of the neurons without inputs and other biases, see Equation 6.4, where the input to any neuron is set to zero. We notice that except for the 1-layer case, the equilibrium value of the neurons is around the middle of the allowed range $[0, 1]$, which enables the neurons to start in a state far from saturation and thus maximizes the number of active neurons. In the case of a single layer, the observed bias is the bias applied to the input and mainly serves to recenter the input value. Since the mean value of the input is positive, the fixed bias needs to act as an additional negative

Table 6.1 – Optimal hyperparameters versus number of layers

number of layers	1	2	3	4
S	0.30	0.37	0.31	0.35
b_{fixed} (GHz)	-0.04	0.18	0.38	0.31
γ (GHz)	0.055	0.088	0.133	0.139
τ_{relax} (ns)	18.2	11.4	7.52	7.19
$x_{equilibrium}$	0	0.51	0.65	0.55

damping.

The optimal damping increases with the number of layers, meaning that the optimal relaxation time of the neurons decreases with the number of layers. This can be understood in terms of dynamics : the input varies quickly in time and needs to be integrated non-linearly by the neurons to produce a slowly varying output signal. When feeding the output of a layer to the next one, the signal is integrated twice. As we need to keep the same output dynamics to classify correctly, each layer needs to have half of the preceding integration time. This can be seen in Figure 6.5, where the responses of neurons are represented for each layer in networks with 1 to 4 layers. The signal is smoothed while progressing through the layers, and the fewer layers the network has, the more each layer needs to integrate the signal. It seems that the supplementary integration effect is only present up to three layers, with the dynamics of the fourth layer being very similar to the third one. In terms of memory, we can also consider that the memory is spread over the layers. This is interesting in terms of hardware application, as we can use oscillators with greater damping, which are easier to produce if we use multilayers.

$$\begin{aligned} \frac{dx}{dt} &= -\gamma x + b_{fixed} \times x(1-x) \\ x_{equilibrium} &= 1 - \frac{\gamma}{b_{fixed}} \end{aligned} \tag{6.4}$$

In Figure 6.6, we display the accuracy for networks with one to four layers versus the input timescale. As designed in the hyperparameters optimisation, the median accuracy – represented in solid lines – is maximal around a time between two inputs equal to one nanosecond for all number of layers. Looking at the mean accuracy around the condition of optimisation, the optimal number of layer is three. Indeed going up to three layers increase the median accuracy around $\delta t = 1ns$ but adding a fourth layer does not improve the accuracy anymore, this is coherent with the admitted optimal number of layers

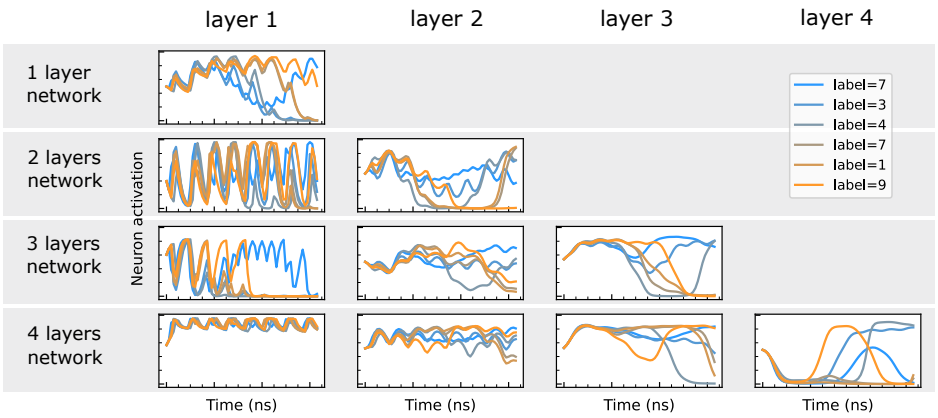


Figure 6.5 – Evolution of neurons from all layers for networks with different numbers of layers when submitted to different inputs. Neurons in the last layers present slower dynamics, which helps to discriminate inputs.

for RNN which is two to three. Here we can observe that the two layer network while having slightly less good performance around $\delta t = 1ns$, is a bit more adaptable at higher time between two inputs. We will show in the section 6.9 that the three layer network can be optimised to present high-adaptability and thus a flat high accuracy on a broad range of input timescales.

In Figure 6.6, the spread of accuracy for 10 different networks initialisation is represented in light colored areas, each initialisation differs only by the random seed used to generate the starting values of weights and biases of the network. There is a great dispersion of performance for a same input timescale : some networks performs well with up to 80% accuracy while others fails with accuracy below 30%. This training failure does not appear in many initialisations : in the case of the three-layer network, only 0.5% of initialisations have an accuracies below 50%. Moreover, we can still reduce this proportion. This issue is due to the saturation of some neurons during the training and will be discussed in the following section.

6.7.2 . Preventing saturation of neurons during training

In order to optimise the network's general behavior, we need to understand and try to tackle the issue mentioned previously : while some initialisations give great results, others with similar parameters fail to train efficiently. This issue is due to some neurons being pushed toward saturation during training, thus changing from an active classification behavior to a passive one. In Figure 6.7, we can see the comparison between the training of a successful network and a failing one. Looking at accuracies through training, we see

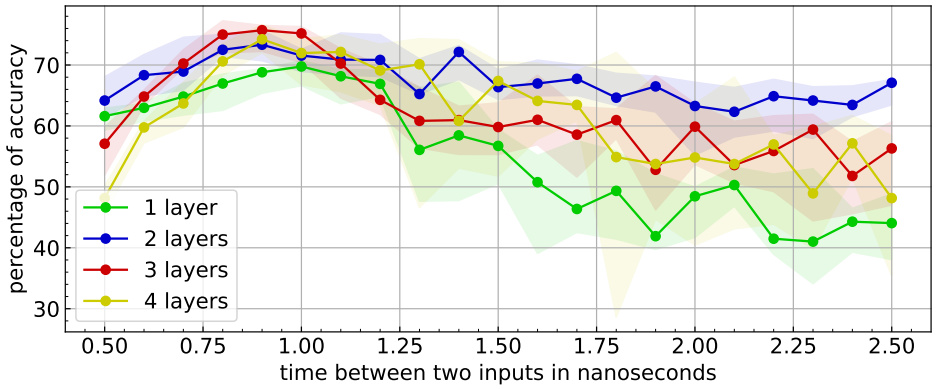


Figure 6.6 – Median accuracy is shown in solid lines, with the range of accuracy in light-colored shadows and the range between the 25th and 75th percentiles in darker-colored shadows for networks with 1, 2, 3, and 4 layers, versus the time between two successive inputs. These accuracies are computed for 10 different network initialisations at each input timescale.

that the successful network's accuracy increases smoothly, while for the failing one, it starts to increase, and a sudden drop in accuracy arises around epoch 24, see Figure 6.7 c) and d). In the meantime, the mean absolute value of trainable parameters, represented in Figure 6.7 a) and b), converges for the successful one and diverges for the failing one. This divergence causes saturation of neurons, driving them too strongly up or down and pushing them into saturated states of 1 or 0.

This is represented in Figure 6.8. Here, plots a), b), and c) show neuron responses to different inputs for three neurons, one from each layer. These neuron responses are taken before the drop in accuracy of a failing network, at epoch 22. In plots d), e), and f), the responses of the same neurons are displayed but taken after the drop in accuracy. We can see that before failing, the neurons of the two last layers show a rich behavior with various trajectories for different inputs. However, after the drop, neurons reach saturation for all inputs, thus losing memory of previous values, which prevents any classification.

A solution to prevent saturation is to maintain lower weight and bias absolute values, which can be done by clipping the weights and biases if they exceed a threshold value. This clipping is rescaled by the input timescale. Indeed, fast-varying signals impose a given input for short times; thus, they need high weight values to drive the transient dynamics of the neurons. The impact of the input timescale and its link to the weight values will be studied in more

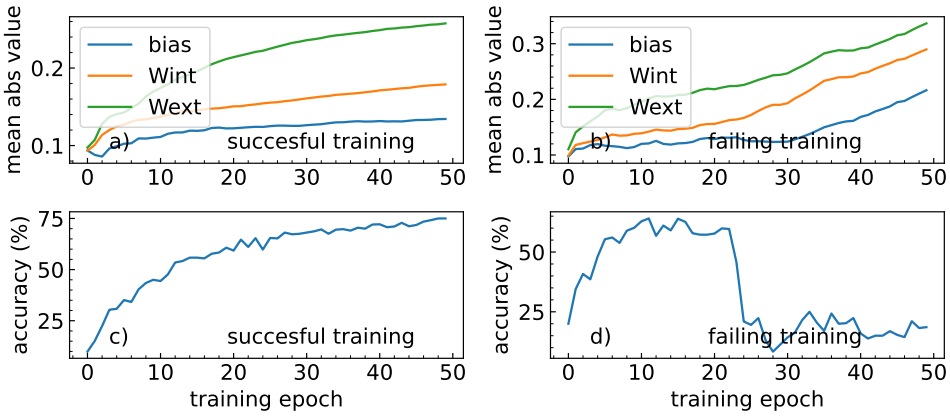


Figure 6.7 – Evolution of the mean absolute value of the different trainable parameters versus the number of epochs for a successful training (a) and failing training (b). Evolution of the test accuracy versus the number of epochs for a successful training (c) and failing training (d).

detail in Section 6.9.

The impact of such clipping is represented in Figure 6.9. The red curve is the accuracy of the three-layer network without clipping, and the blue curve is the same network trained with weights clipped to $\frac{0.25}{\delta t}$ in absolute value and biases clipped in the same way to $\frac{0.15}{\delta t}$. We see that it solves the issue of large discrepancies : all initialisations now train well. However, the maximal accuracy is reduced. Without clipping, 75% of the networks were performing better than 70% accuracy, and 50% better than 75% at $\delta t = 1,$ ns, but now the accuracy is around 67% for all networks. In the meantime, the time adaptation is more uniform; the accuracy is quite flat between $\delta t = 0.75$ns and $\delta t = 2$ns. This can be understood as shorter timescales needing higher weight values to drive neurons strongly for a short time; the degradation comes from the impossibility of reaching such a strong drive. On the other hand, at longer timescales, excessively large weights are more prone to saturate neurons, which is now prevented with the clipping, thus improving median accuracy.

6.7.3 . Hyperoptimisation with Optuna

In this section, we demonstrate that it is possible to obtain high accuracy and high input timescale adaptation with a more refined procedure of hyper-optimisation. The hyper-optimisation is realized by adding six additional hyperparameters. The learning rate, the learning rate decay, the density of intra-layer connections, and the density of inter-layer connections are now hyperparameters. Additionally, the input $I_0(t)$ is rescaled as $S_{in} \times I_0(t) + b_{in}$, with S_{in} and b_{in} being also hyperparameters. A first round of training is perfor-

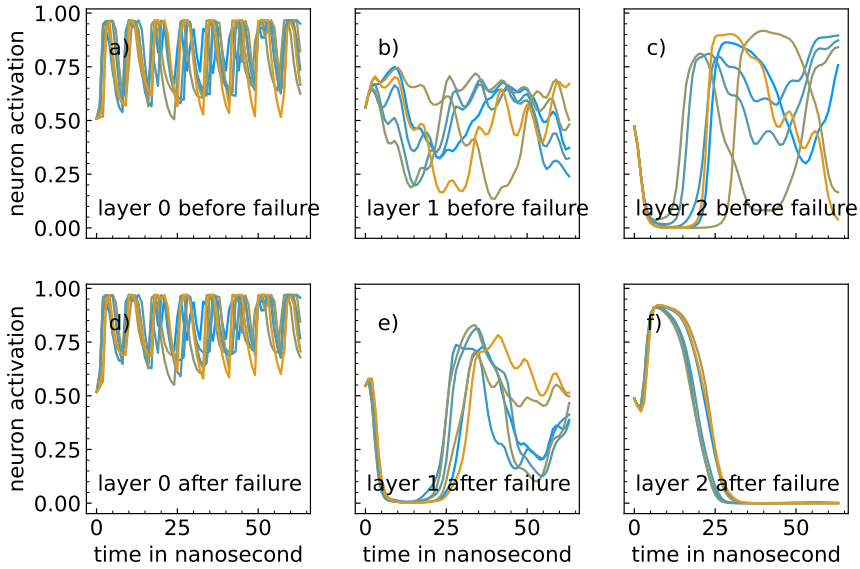


Figure 6.8 – Evolution of neuron responses versus time when submitted to different input examples. Neuron from layer 0 before failure of training in (a). Neuron from layer 1 before failure of training in (b). Neuron from layer 2 before failure of training in (c). Neuron from layer 0 after failure of training in (d). Neuron from layer 1 after failure of training in (e). Neuron from layer 2 after failure of training in (f).

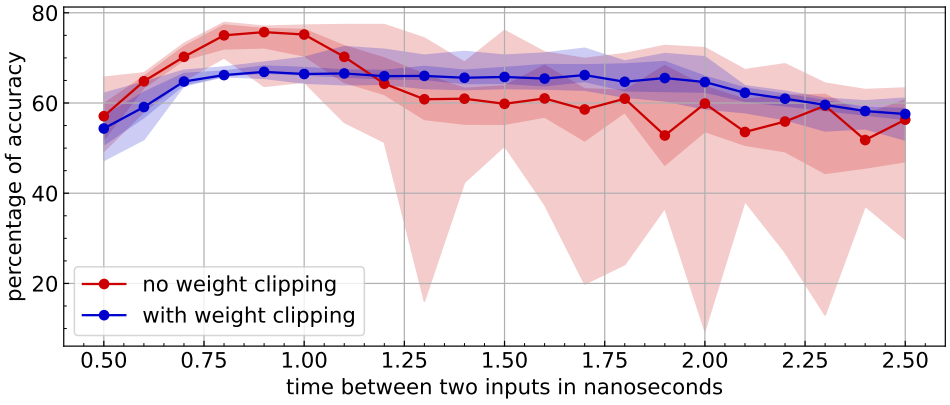


Figure 6.9 – Accuracy of networks versus time between two input points. For each input timescale, the accuracy is computed for 10 different initialisations. The median is displayed in solid lines, the range of accuracy in light-colored shadows, and the range between the 25th and 75th percentiles in darker-colored shadows. These quantities are computed for a network with weight clipping in blue and without clipping in red.

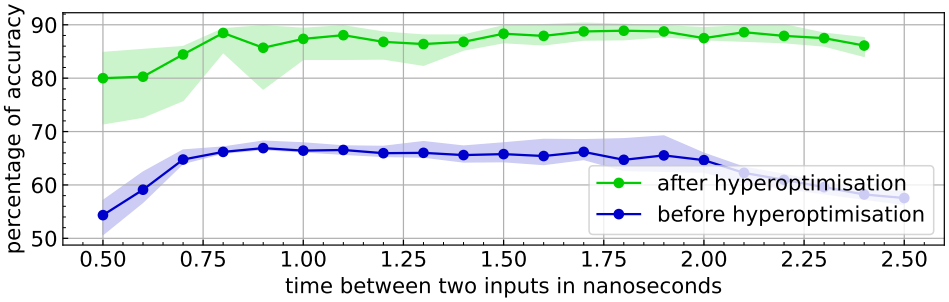


Figure 6.10 – Accuracy of hyper-optimised networks versus time between two inputs points. For each input timescale, the accuracy is computed for 36 different initialisations. The median is displayed in solid lines, and the range between the 25th and 75th percentiles is shown in colored shadows.

med with 120 random hyperparameter configurations and no pruning. After identifying the more promising region, a second round of training is performed with optimisation activated and pruning to find the best hyperparameter configuration.

After this hyper-optimisation step, the performance of the network is globally increased as displayed in Figure 6.10. We see that we now have a highest performance of 93.89%, and the top ten best performances range from 93.06% to 93.89%. Looking at the median accuracy and above the 25^{th} percentile, the accuracy is quite flat, meaning that our network is adapting to the input timescale around its hyper-optimisation point. However, some networks are now failing in training again; here, it's only a small proportion, only 7.2% of the initialisations lead to less than 70% accuracy. To ensure full training, we could perform a fine-tuning of the weight clipping that is no longer valid; indeed, some hyperparameters such as S_{in} or S have new values that change the strength of the drives.

Hyperparameters have different levels of importance and are more or less constrained. The amplification factor between layers appears to be a strongly constrained parameter. In Figure 6.11, I varied the amplification factors individually—the input amplification S_{in} and interlayer amplification S—and simultaneously, around their optimal value. The top accuracy above 85% is achieved in a small window when varying the interlayer amplification S; this parameter can be varied only by a factor of 1 to 2 to obtain high accuracy and thus must be tuned carefully. This amplification factor controls how much signal is transmitted from one layer to the next. If the input signal to a layer is too weak, for instance with $S = 0.0315$, neurons will have a weak response and stay around their equilibrium position $x \approx 0.5$, as we can see in Figure

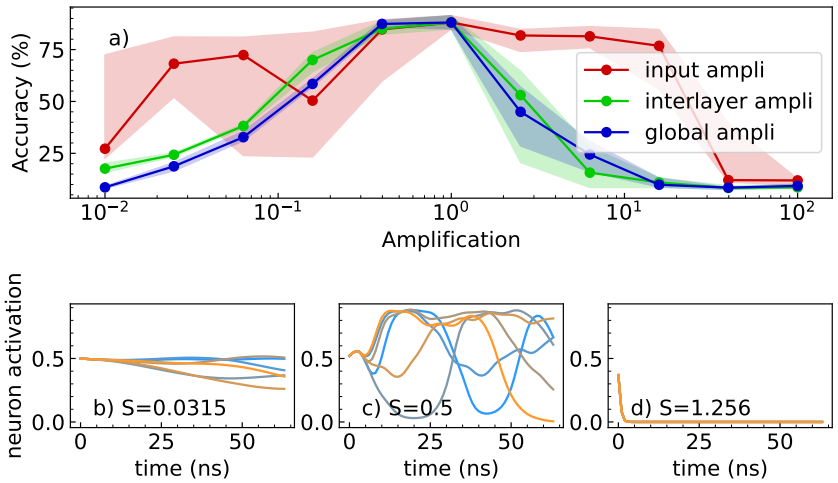


Figure 6.11 – a) Median accuracy of 10 different initialisations of networks while varying the different amplification factors : input and interlayer amplifications. b), c), and d) Responses of neurons from the last layer for different global amplification factors.

6.11 a). Similarly, if the signal is too strong, neurons will be rapidly driven into saturation states, either zero or one, as we can see with $S = 1.256$, see plot c). These two situations lead to nearly inactive neurons that don't have a rich trajectory and thus cannot perform classification. We can identify three different regimes : underdriven neurons, active neurons, and overdriven neurons.

The learning rate and the relaxation time impact will be studied more in depth in Section 6.9; these two quantities are loosely constrained and can vary by a factor of 1 to 10 while still giving optimal results. Biases need to compensate for the damping and thus share the same constraints as the relaxation time.

6.8 . Reducing the density of connection while maintaining high-accuracy

When performing the hyperoptimisation, the density of connections at initialisation between oscillators didn't appear as a critical parameter; some runs with low density were able to attain high accuracy. I thus tested how sparse the network can be at initialisation while still showing high accuracy after training. For this, I scanned the interlayer density of connections and intralayer density of connections both individually and simultaneously. Results are displayed in Figure 6.12. We can observe that when keeping the interlayer

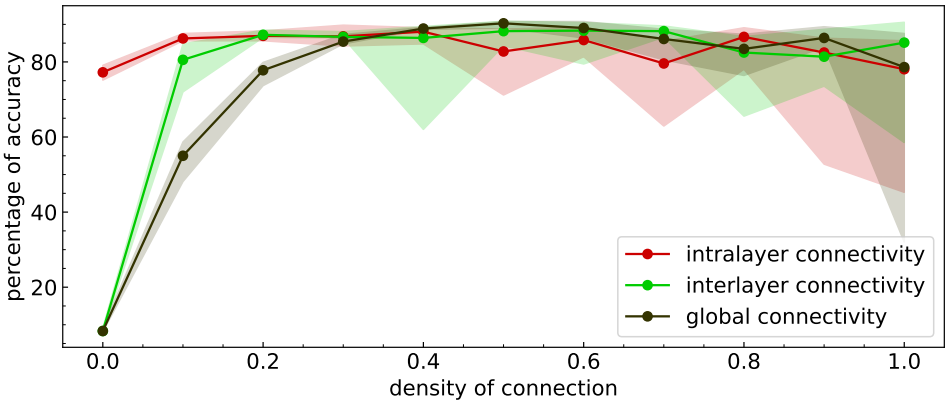


Figure 6.12 – Evolution of the test accuracy when reducing the density of connectivity within a layer and between layers. Each point represents 10 different random initialisations.

density of connections at 100%, the intralayer density of connections can be drastically reduced down to 10% without loss of accuracy and to 0% with a loss of 10% accuracy. Similarly, keeping the intralayer density and reducing the interlayer density down to 20% doesn't degrade the accuracy. Below 20%, the accuracy decreases and ultimately hits 10%, corresponding to random classification when the interlayer density is set to 0%. This is expected as the input signal is no longer flowing through the network. Varying both densities simultaneously leads to a maximum accuracy at 50% connectivity, and we can go as low as 30% connectivity without significant loss of accuracy.

Another effect of the reduction of connectivity can be observed in Figure 6.12. It appears that reducing the connectivity also reduces the accuracy dispersion of the different initialisations. Here, for each density step, 10 random initialisations are trained. When the global density is at 100%, the dispersion, visible by the 75th to 25th percentile range represented in colored shadows, is quite large. On the contrary, when the global density of connections is below 60%, the dispersion is small and continues to decrease when decreasing the connectivity. Diminishing the connectivity reduces the number of inputs going to each neuron; this also decreases the risk of saturating neurons during the training, which explains the smaller dispersion—fewer networks face the issue of saturating neurons during training.

This reduction of connectivity is promising for hardware implementation as it reduces the number of required synapses and connections. This pre-training random pruning could also be combined with pruning of low-importance connections during training.

6.9 . How to adapt a spintronic network to the input timescale

In this section, we aim to investigate how the spintronic network adapts itself to different input timescales. The network's hyperparameters have been optimised to obtain the best accuracy for a time of one nanosecond between two successive input points from an image. We then train networks with these hyperparameters for a range of different times between two inputs. As we train the dynamics of the network, it is able to adapt to a broad range of input timescales, meaning that it is able to adapt to fast or slow varying input signals.

6.9.1 . Ensuring non-saturated oscillators at all input timescales

In order to obtain a network able to process the widest range of input timescales, we need to ensure that neurons aren't saturated in their steady-state. Previously, the input in the first layer was not centered on zero, and thus the fixed bias of this layer was there to both compensate for the mean value of the input and the damping. Similarly, the fixed biases for layers two and three had to compensate for the damping. The values were found via optimisation ; however, this approach fails when the input timescale is changed by orders of magnitude. Indeed, an ill-compensated damping or mean value can lead to a steady-state of the neurons near saturation points, suppressing the time-oscillation from the input.

To avoid this issue, the input is now centered around zero by subtracting its mean value ; this could also be achieved via a high-pass filter. The fixed bias of each layer is set to two times the damping. By choosing this value and in the absence of input, the evolution equation of a neuron becomes Equation 6.5. Here, we see that $x_i^{n+1} = 0.5$ is a stable equilibrium point. When a time-varying input centered on zero is applied, it will drive the neuron to oscillate around $x_i^{n+1} = 0.5$, limiting the risk of saturation.

$$\begin{aligned} \frac{dx_i^{n+1}}{dt} &= -\gamma x_i^{n+1} + 2\gamma x_i^{n+1}(1 - x_i^{n+1}) \\ \frac{dx_i^{n+1}}{dt} &= 2\gamma x_i^{n+1}(0.5 - x_i^{n+1}) \end{aligned} \tag{6.5}$$

All following developments are performed in this configuration, where $b_{in} = b_{fixed} = 2\gamma$ and the input is centered around zero. We can now observe the impact of the main parameters on the training of the network.

6.9.2 . Impact of varying dynamics parameters of the network at

fixed input timescale

Two parameters are of particular importance for adaptation to the input timescale : the learning rate and the relaxation time of the oscillators. The learning rate, similarly to the amplification factor S, defines the strength of the input drive. The relaxation time is linked to the memory of the neuron.

I scanned the learning rate around its optimal value of 0.0148 and the results are displayed in Figure 6.13. The learning rate can be varied by a factor of 1 to 10 while keeping maximal accuracy. Moreover, we can look at the neurons' behavior when the learning rate is smaller or bigger than the optimal value. Neurons' responses to several inputs are displayed in Figure 6.13.

When the learning rate is too small, $lr = 10^{-4}$ for instance, neurons present smoothed-out trajectories ; this is similar to the underdriving regime observed when varying the amplification between layers. Neurons in this case react too weakly and slowly to the fast-varying input because weights aren't high enough to drive the neurons strongly. This causes a strong time-averaging of the input signal and thus degrades the ability of the network to discriminate between inputs. Additionally, over-averaging is more prone to push neurons to saturation as neuron trajectories present fewer oscillations, which is also a cause of degradation of the accuracy. This saturation due to over-averaging is visible in Figure 6.13.

When the learning rate is too high, $lr = 10^{-1}$ for instance, neurons are immediately saturated. This is the overdrive regime observed with an amplification factor that is too high between layers. In this case, the learning rate is updating the weights to high values that push the neurons to saturation. We can notice that while when the learning rate is small, $lr = 10^{-4}$, 0% of the weights are clipped, and when the learning rate is optimal, $lr = 10^{-2}$, only around 10% are clipped, when the learning rate is equal to $lr = 10^{-1}$, 100% of the weights are clipped, also preventing learning.

Finally, it is interesting to note that having a too-small learning rate seems to lower the accuracy less drastically than a too-large learning rate value ; over-averaging is less critical than overdriving.

I scanned the relaxation time around its optimal value of 14.12 ; results are displayed in Figure 6.14. The relaxation time can be varied by a factor of 1 to 10 while keeping maximal accuracy. Moreover, we look at the neurons' behavior when the relaxation time is smaller or bigger than the optimal value. Neurons' responses to several inputs are displayed in Figure 6.14.

When the relaxation time is greater than the optimal one, we notice a slow decrease in accuracy. This decrease is due to the damping being slightly too small, not pulling the neurons strongly enough toward $x = 0.5$. As a result, the weights pull the neurons slightly too much, leading to a slightly overdri-

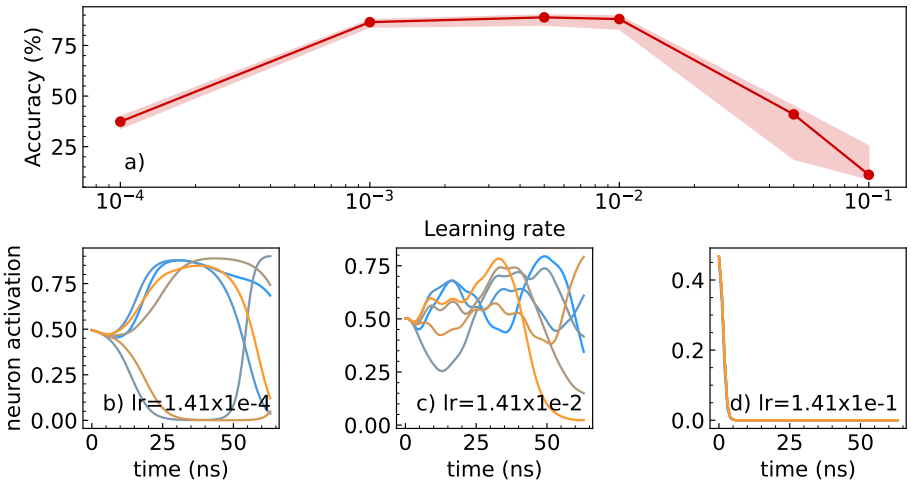


Figure 6.13 – Test accuracy versus learning rate. For each value of the learning rate, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the red shadowed area corresponds to the limits of the 25th and 75th percentiles. Graphs a), b), and c) represent the response of a neuron from the last layer of the network submitted to different inputs examples, represented by colors from blue to orange. The three graphs correspond to different learning rates.

ven system with oscillators sometimes reaching saturation and thus losing memory of the previous input values. However, the training tends to keep the weights small to balance this overdriving due to the small damping. Indeed, when the relaxation time is optimal, at $\tau = 14.12$, the mean value of the interlayer weight is 0.20, and when the relaxation time is $\tau = 1412.00$, the mean value of the interlayer weight is 0.14. This balance is possible here because the learning rate is reasonably low, $lr = 1.4 \times 10^{-2}$; thus, even with a hundred times longer relaxation time, the accuracy can stay quite high. We can also notice that the low damping causes the neurons to over-average the inputs in time, resulting in too smooth and simple trajectories that reduce the classification power of the network.

When the relaxation time is smaller than the optimal one, we can notice a sharp decrease in accuracy. In this case, the damping is too strong; it pulls the neurons too strongly toward the equilibrium point $x = 0.5$. This overdamping regime resembles the underdriving regime, where neurons struggle to have complex and rich trajectories. The weights are nearly all hitting their clipping value for $\tau = 10^{-1}$; higher weights would be needed to compensate for the strong damping and drive the neurons away from their equilibrium point. In Section 6.9.3, we will also investigate the relation between the relaxation time of the oscillators and the timescale of variation of the input.

In this section, we observed the efficiency of training while varying the learning rate or the relaxation time of the oscillators. Both parameters present a broad range of acceptable values, with a ratio of approximately 1 to 10 between the minimal and maximal acceptable values. These scans have been performed at a fixed input timescale; however, we already noticed that oscillators need to have a driving term $S \times W \times I(t)$ that is comparable to the damping γ. Indeed, if the damping is too strong compared to the drive, neurons are underdriven, staying around their equilibrium point $x = 0.5$; if the drive is too strong compared to the damping, neurons are overdriven and hit saturation. Another effect that we observed is time over-averaging, which happens when the drive or damping are smaller than their optimal values. In this case, neurons react too slowly to the variation of the input. This will be discussed more in depth when varying the input timescale in the next section 6.9.3.

6.9.3 . Impact of the network parameters on input timescale adaptation

We observed the response of the network at a fixed input timescale; however, the speed of variation of the input also plays an important role in the response of the neurons. In this section, we will vary the time between two consecutive inputs and record the accuracy of networks trained with this input timescale. In Figure 6.15 a), the final test accuracy is displayed, varying the

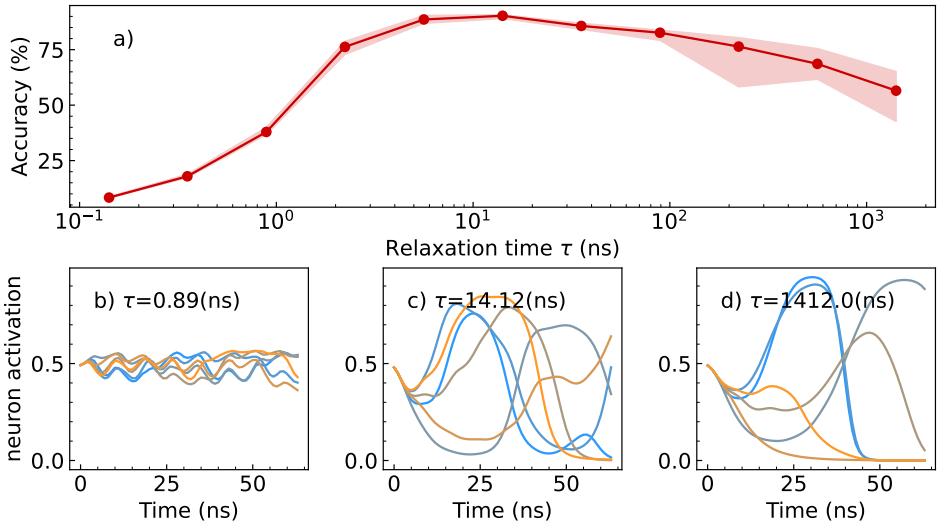


Figure 6.14 – Test accuracy versus relaxation time of the spintronic neurons. For each value of relaxation time, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the red shadowed area corresponds to the limits of the 25th and 75th percentiles. Graphs a), b), and c) represent the response of a neuron from the last layer of the network submitted to different inputs, represented by colors from blue to orange. The three graphs correspond to different relaxation times.

input timescale from $\Delta t = 10^{-2}$, ns to $\Delta t = 10^{2}$, ns, where the network hyperparameters have been optimised for $\Delta t = 1$ns. At each input timescale, ten networks with different initialisations are trained. It is important to note that the weights are clipped to a max value rescaled by the input timescale $w_{clip} = \pm\frac{w_{max}}{\Delta t}$. A first interesting result is that our optimisation procedure is working, the top accuracy is obtained for $\Delta t = 1$; moreover, the input timescale presents an optimal window around this top accuracy with a range of approximately 1 to 10, from $\Delta t = 0.38$ns to $\Delta t = 2.51$ns.

In Figures 6.15 b), c), and d), the response of a representative neuron from the last layer of the network after training is presented, submitted to different input examples. These three graphs differ in the input timescale with which the network have been trained and tested. In graph c), the input timescale is the optimal one $\Delta t = 1$ns, and we observe that the neuron presents a rich dynamics with different trajectories for different inputs, enabling good classification. In plot b), the input timescale is $\Delta t = 0.06$ns, meaning that the input is varying very fast compared to the relaxation time of the oscillators $\tau = 14.12$ns. In this case, the neuron has less complex trajectories; they appear more smoothed out and similar for different inputs. This is coherent with the lower accuracy compared to the previous case. There are two causes for these smoothed-out trajectories : first, neurons, due to their long relaxation time scale, tend to average out the inputs; however, as shown in Section 6.4, the inputs cannot be correctly classified based only on their mean values. The second cause is that weights are too small to drive the neuron sufficiently when the input signal varies rapidly. These two causes will be untangled more in depth in the following. In graph d), the neuron response is displayed for an input timescale of $\Delta t = 39.81$ns, around three times the relaxation time of the neurons. In this case, we observe that the neuron response is kept around $x = 0.5$ with very small oscillations. From examining the weights values, we notice that 100% of them are clipped to the clipping value, meaning that the training pushed the weights to their maximal values without managing to drive the neuron strongly enough. To understand the system behavior more clearly, we need to release part of the constraint on the weights; for this, I repeated the same input timescale scan but with a uniform clipping $w_{clip} = \pm w_{max}$.

The results of this new input timescale scan are displayed in Figure 6.16. The top accuracy is not modified as expected; the accuracy for fast-varying inputs is slightly lower. This can be understood as the weights are now hitting the clipping that was previously higher. Let's now focus on the region of large input timescales. The accuracy is slightly lower and more spread, which is due to some neurons saturating now that the clipping is higher. In graph d), the response of a neuron from a network trained with $\Delta t = 39.81$ns is displayed.

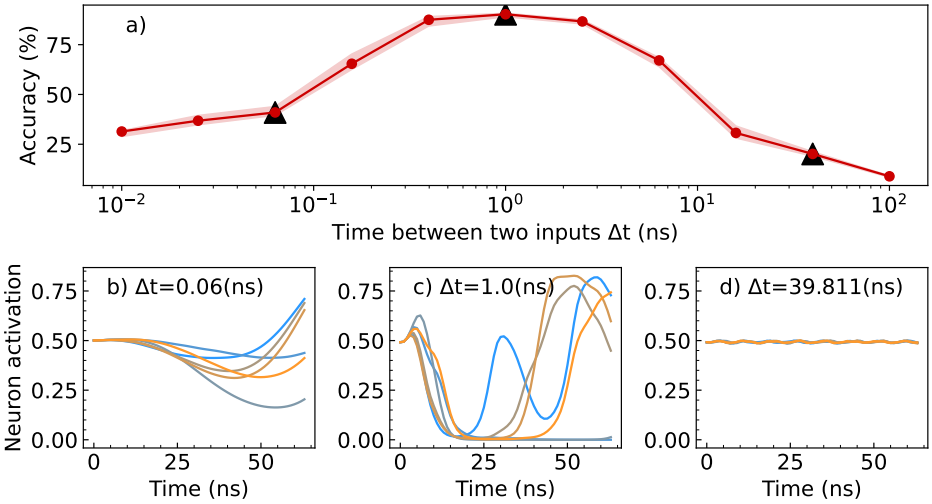


Figure 6.15 – Test accuracy versus input timescale Δt, corresponding to the time of application of a point from a time series. For each value of the input timescale, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the red shadowed area corresponds to the limits of the 25th and 75th percentiles. Graphs a), b), and c) represent the response of a neuron from the last layer of the network submitted to different inputs, represented by colors from blue to orange. The three graphs correspond to different input timescales.

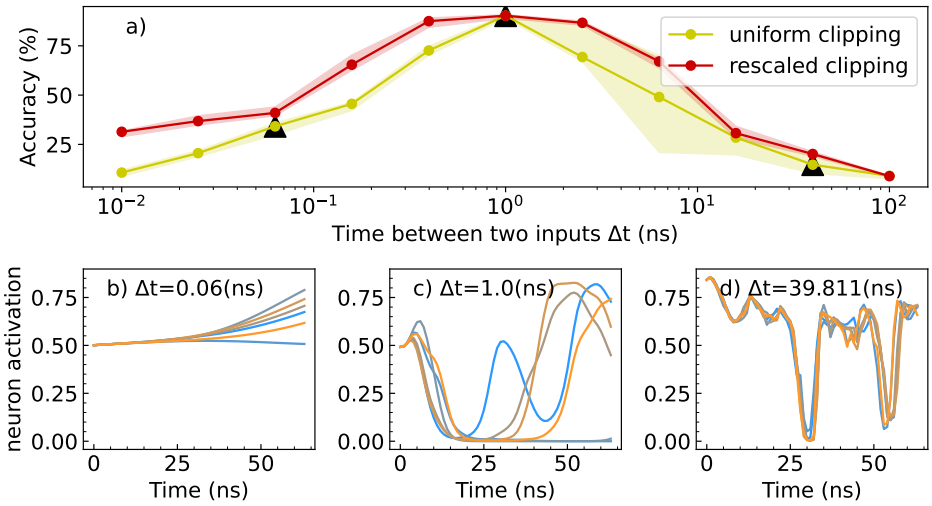


Figure 6.16 – Test accuracy versus input timescale Δt, corresponding to the time of application of an input point from a time series. For each value of the input timescale, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the shadowed area corresponds to the limits of the 25th and 75th percentiles. The accuracy is displayed in two cases : in red, the weights are clipped with a maximal value that depends on the input timescale ; in yellow, the weights are clipped with a maximal value independent of the input timescale. Graphs a), b), and c) represent the response of a neuron from the last layer of the network submitted to different inputs, represented by colors from blue to orange. The considered networks are those trained with a clipping uniform in input timescale. The three graphs correspond to different input timescales.

Since the clipping is higher, higher weight values are applied on the input, allowing the neuron to be driven out of equilibrium. However, the trajectories appear to be very similar to each other and ill-suited to discriminate between inputs. This effect can be explained by the absence of sufficient memory in the system ; here, the relaxation time of the neurons is three times smaller than the time between two different inputs, meaning that nearly all previous inputs are forgotten when sending a new one. The neuron is not integrating the input but only reaching its steady state depending on the current inputs.

In order to test the validity of my interpretations of the neurons' behavior and the link with the final observed accuracy, I performed the same input timescale scans but now varying the relaxation time and learning rate ; results are displayed in Figure 6.17 a) and b).

If we look at the region of small input timescales, it appears that having a smaller relaxation timescale can slightly improve the accuracy, as shown

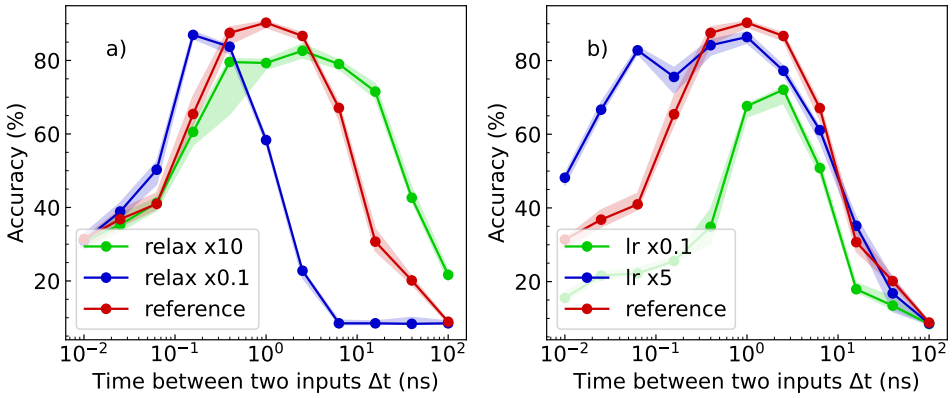


Figure 6.17 – Test accuracy versus input timescale Δt, corresponding to the time of application of an input point from a time series. For each value of the input timescale, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the shadowed area corresponds to the limits of the 25th and 75th percentiles. Graph a) displays the test accuracy for networks trained with the reference relaxation time $\tau = 14.12$ ns, relaxation times ten times smaller in blue, and ten times bigger in green. Graph b) displays the test accuracy for networks trained with the reference learning rate $lr = 0.014$, and learning rates ten times smaller in green and five times bigger in blue.

in graph a). This is in agreement with the over-averaging interpretation. In this case, a lower relaxation time reduces the time-averaging of the neurons. However, the main parameter to improve accuracy at small input timescales is to have a larger learning rate, as visible in graph b). This is coherent with the issue of underdriving the neurons, as a higher learning rate gives larger weights able to drive the neuron in its transient state.

Focusing now on the large input timescale region, we can notice from graphs a) and b) that only increasing the relaxation time, and thus the memory length of the system, can improve the accuracy in this region.

Now that my interpretations are validated, I propose a metric in Equation 6.6 to evaluate if a spintronic network is likely to perform efficiently with given parameters τ, S, lr, and Δt. This equation states that the damping $\frac{1}{\tau}$ must be approximately equal to the driving term. The driving term is composed of the amplification factor, the weights that depend themselves on the learning rate, the input signal, and the ratio of its varying timescale Δt to the relaxation time of the neurons. This last term represents the fact that if a drive is varying fast compared to the relaxation time, it gets averaged by the neuron. Ideally, the learning rate must be a fraction of the final weight value, between $\frac{1}{10}$ and $\frac{1}{100}$. An additional condition is that the relaxation time is longer than the input

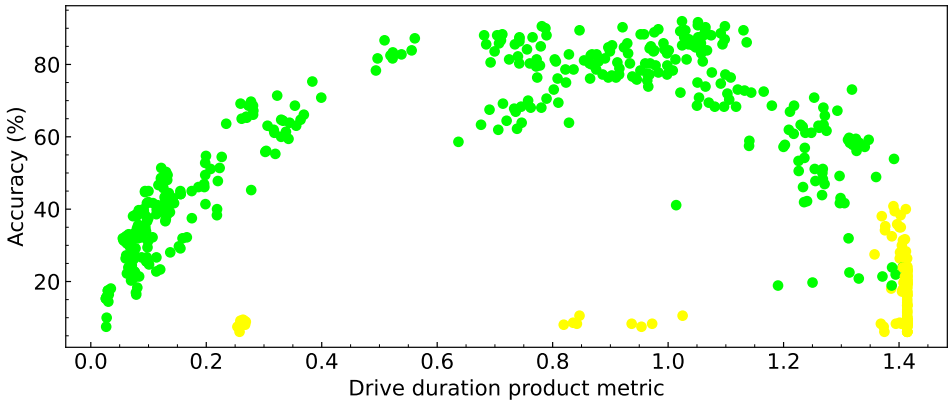


Figure 6.18 – Test accuracy versus drive duration product metric for all networks displayed in figure 6.17. Networks respecting the condition $\tau > \Delta t$ are displayed in green, the other ones in yellow.

timescale in order to retain memory of the previous inputs.

$$\begin{aligned} &\frac{1}{\tau} \approx SW_{lr} \times \frac{I(t)\Delta t}{\tau} \\ &1 \approx SW_{lr} \times I(t)\Delta t \\ &\text{with } \tau > \Delta t \end{aligned} \tag{6.6}$$

I evaluate this metric on all the trained networks displayed in Figure 6.17. Here, the input is equal to $\sqrt{32 \times 2}$ as each neuron receives the inputs from 2 times 32 other neurons with an output of amplitude 1. The weight value is estimated by taking the mean absolute values of the weights. The results are displayed in Figure 6.18. We observe that the successful networks are indeed centered around a metric equal to 1. Networks not respecting the relaxation time condition are displayed in yellow and fail to have a high accuracy.

6.9.4 . Comparison with a standard recurrent network (CTRNN)

The system being well understood and optimised, I compare its performance with a standard continuous time recurrent neural network (CTRNN) with a similar structure. In this case, the neurons obey the equation 6.7. These neurons have non-bounded internal values x_i^n, i.e. these internal values aren't clipped. The neurons don't present liquid time constants. A hyperbolic tangent function is applied to the output of each neuron, which is a standard activation function procedure in RNNs. This function also limits the drives applied on neurons and reduces the risks of fast divergences. The relaxation time $\tau = \frac{1}{\gamma}$ is designed to recenter the neurons' outputs and prevent divergences. This network is evaluated with the same hyperparameters as the spintronic network,

including relaxation time and amplification. The only parameter that changes is the learning rate, which is divided by 4. This change is made because, in the spintronic case, the input is multiplied by $x(1-x) \approx 0.5 \times 0.5 = 0.25$; thus, the drive in the CTRNN requires four times smaller weights to be comparable. The result of the input timescale scan is displayed in Figure 6.19. The mean accuracy of the CTRNN is 89.00 $\pm$ 3.48 $\%$, similar to the spintronic network performance of 89.83 $\pm$ 2.91 $\%$. The time-adaptation window is also very similar. The slightly superior performance at large input timescales can be explained because spintronic neurons are bounded, which is not directly the case in the CTRNN. It's also possible that the weights are slightly smaller in the CTRNN case, thus shifting the optimal accuracy to higher input timescales. With our simple task, the spintronic network doesn't present time-adaptation benefits from its liquid time constant compared to a CTRNN. However, I demonstrated that a simulated network of connected spintronic neurons can perform as well as a pure software recurrent network despite the strong constraint of using bounded neurons. This is encouraging for building hardware networks using transient device dynamics, as physical devices have naturally bounded outputs.

$$\begin{aligned} &\frac{dx_i^{n+1}}{dt} = -\gamma x_i^{n+1} + I(t) \\ &\text{with } I(t) = S \times ((W_{ext}^n)_{ij} y_j^n + (W_{int}^{n+1})_{ij} y_j^{n+1} + b_i^{n+1}) + b_{fixed} \\ &y_j^n = tanh(x_j^n) \end{aligned} \tag{6.7}$$

6.10 . Hardware perspectives

From this simulation study, we can extract some guidelines for the realization of a hardware network made of dynamical neurons. Assuming that the timescale of the input is known, neurons should be chosen with a relaxation time larger than this input timescale, ideally around 10 times larger. The network must be trained using a learning rate that will lead to weights being of the same order of magnitude as the neuron damping. Additionally, in order to have active neurons, a constant bias needs to be sent to the neurons; in general, this bias needs to be bigger than the damping, about two times in the spintronic case.

Between two series of inputs, such as two digits in our case, the network must relax to its original state. For this, we can just remove all inputs and wait a few times the relaxation time of the oscillators.

A multilayer network can have better performance, but it will also have different dynamics. As seen in the previous section, the optimal relaxation time is shorter in a multilayer network than in a monolayer network. If the

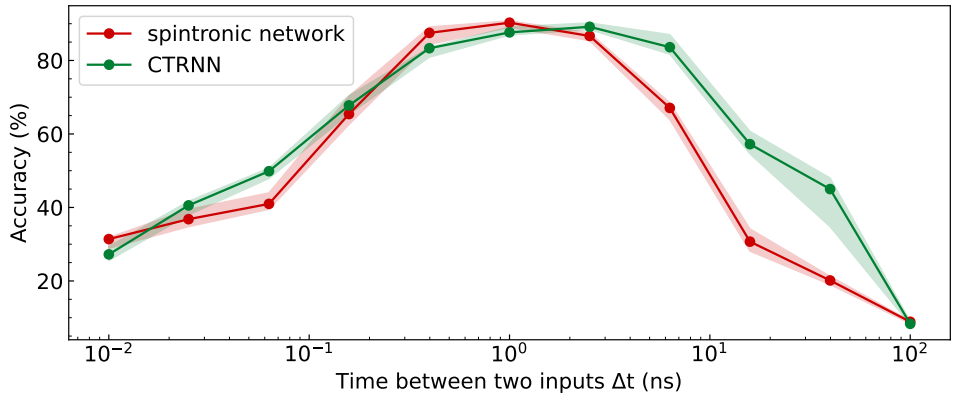


Figure 6.19 – Test accuracy versus input timescale Δt, corresponding to the time of application of an input point from a time series. For each value of the input timescale, 10 different network initialisations are trained and tested. The solid line displays the median accuracy, and the shadowed area corresponds to the limits of the 25th and 75th percentiles. The accuracy is displayed in red for spintronic networks and in green for continuous-time recurrent networks.

physical neurons have too short a relaxation time for a single-layer network, it is then possible to use them in a multilayer network. However, chaining layers requires implementing a high-pass filter to recenter the inputs to the layers following the first one. This can be realized with a capacitor if the outputs of the neurons are DC electrical signals.

A spintronic network using dynamical neurons would require synapses that transform RF power into DC voltages, these DC voltages being then filtered through a capacitor acting as a high-pass filter. This synaptic behavior has already been demonstrated in MTJ [83][86][80] and with simple Permalloy devices in previous chapters. However, the dynamics of the synapses should not mix with the dynamics of the neurons; ideally, synapses should react much faster than the neurons. Another possibility would be to also take into account the impact of the dynamics of synapses, which would require another in-depth study. It is interesting to note that the possibility of reducing the density of connections would reduce the number of synapses and thus the devices needed, facilitating the experimental realization of a compact hardware network.

6.11 . Conclusion

In this chapter, I demonstrated that a multilayer spintronic network trained with backpropagation through time can solve a complex task : recognizing handwritten digits. Moreover, I showed that adding layers improves the

performance. Several issues linked to the chaining of layers had to be considered both for the simulation and for a future hardware implementation, such as the filtering of output offsets with high-pass filters and the neurons' saturation. Through hyper-optimization, we can obtain a network with high performance, achieving more than 90% accuracy over a wide range of input timescales, demonstrating the time adaptation of the network through training of the neurons' driving strength. The link between the dynamics of the input and neurons' relaxation has been studied, and I built a metric to estimate if a network is likely to be successful ; this metric evaluates the matching between weights and the speed of variation of the input. Moreover, I established that the relaxation time of the neurons has to be smaller than the variation-time between two inputs, around one-tenth to produce the best accuracy.

While the adaptation of the dynamics is well described and understood, the prevention of neurons' saturation could potentially be improved and redesigned as a form of penalty in the loss for weights saturating neurons or by implementing a BPTT with constraints on the neurons' values, as in optimal control theory [117].

These results have been compiled in an article, for which I am the first author, and the preprint is available on arXiv : https ://arxiv.org/abs/2408.02835

Conclusion

In this thesis, we developed spintronic radio-frequency neural networks under two aspects : the experimental realization of a CNN and the simulation of a network with dynamical neurons. To implement a spintronic convolutional layer, I developed AMR spin diode synapses and demonstrated that chains of such synapses can be fabricated with the correct weights. These chains were then integrated into a hybrid hardware-software CNN, and this network was able to reach the expected accuracy while taking the noise into account during off-chip training. This opens the path for more complete spintronic neural networks. Meanwhile, I designed the architecture of a multilayer neural network made of coupled STNOs. The goal of this network was to train and exploit the transient dynamics of these devices to classify time series. The matching between devices' characteristic relaxation time, input timescale, and network parameters was investigated. Guidelines on these quantities were then derived to define when a network of dynamic neurons can learn to solve a time-dependent task. Below, we recapitulate in more detail the most important results obtained.

In chapter 2, we showed that AMR spin diodes can be used as radio-frequency synapses. We proposed and tested three different types of synapses. The first one is made of a single spin-diode ; its weight is tuned by the external magnetic field, which is non-volatile in the case of permanent magnets but sensitive to external perturbations. The second one is similar, but the weight is tuned by applying a current in a stripline on top of the diode. This approach is volatile, and the presence of a stripline creates capacitive leakage of the RF inputs, which is an issue in a more complex system. Finally, we proposed and made a design with two opposed spin diodes ; the synaptic weight can be controlled by shifting the relative position of the two diodes. This differential measurement configuration gives rise to a better-defined signal and is robust to external perturbations.

In chapter 3, we realized chains of synapses to implement weighted sums on frequency-multiplexed inputs. A mixed series-parallel configuration was designed to ensure a fixed load of 50 ohms suited for upscaling. A model of a chain response was built based on the calibration with high precision of a single synapse. This model allowed us to predict the chain geometry to implement desired weights. Three chains were fabricated with this procedure, and the resulting weights were in good agreement with the theoretical ones, with an error of 5.2 $\%$ of the reference value of a weight equal to 1.

In chapter 4, we realize a hybrid hardware-software convolutional network able to solve an image classification task using the FashionMNIST dataset. The network is trained in pure software, including some random noise. The

weights obtained after training are fed into the optimization procedure described in chapter 3, which finds the best geometry for three chains to implement the convolutional layer. After fabrication, these chains are placed into the experimental setup, where an automatic alignment procedure is performed to obtain the best matching between experimental and theoretical weights. This network is able to reach an accuracy of 88% on the first one hundred images of the test dataset, compared to the 88.4% obtained with the software including noise.

This first demonstration of the successful application of an experimental spintronic neural network on a complex task opens the path for more intricate networks. The new challenges to address include the efficient co-integration of spintronic neurons and synapses, upscaling to tens or hundreds of synapses, and the realization of networks with nanometric size devices. More development and investigation into the limits of input power and frequency width of synapses are required to achieve these goals. Such a network would benefit from the low power consumption of spintronics and high compactness thanks to the frequency connectivity.

In chapter 5, we demonstrate that a simulated network made of a single layer of coupled STNOs can be trained via backpropagation through time. The targeted task is the discrimination between sine and square waves, which requires non-linearity and memory. We observed that a network with its dynamics trained through backpropagation presents better performance than the same network trained with reservoir computing methods. We can reach 100% accuracy with as few as 2 oscillators when trained through backpropagation, compared to a minimum of 16 oscillators required in reservoir computing. Furthermore, the type of neuron differential equation needed to learn in such systems has been studied, and naturally bounded systems appear to learn better than neurons subjected to artificial clipping. Physical models are thus natural candidates to implement networks of coupled dynamical neurons.

In chapter 6, the architecture of the previous neural network is adapted to build a multilayer network. This network is trained to classify handwritten digit images from the DIGIT dataset sent pixel by pixel. The optimal number of layers is assessed, and a three-layer network gives the best performance. This network is hyper-optimized and reaches a good mean performance of 89.83 $\pm$ 2.91 % accuracy, compared to the 89.00 $\pm$ 3.48 % of a standard CTRNN. The impact of the parameters controlling the dynamics has been studied. The network maintains high performance over a five-fold range around the optimal input timescale for a given relaxation. We also extracted guidelines to match the devices' parameters and the input timescale to permit learning. The re-

laxation time of oscillators must be greater than the input timescale, and the cumulative drive applied to a neuron should be close to one. These guidelines could be useful for future hardware applications or even investigations of networks made of different dynamical neurons. Another observation of interest for hardware applications is that the network can be sparsified down to 50 $\%$ without degrading the accuracy.

In these two chapters, we demonstrated that a network made of coupled STNOs with trainable transient dynamics can solve complex tasks, such as time series classification. Moreover, a multilayer architecture can be realized in a hardware-friendly way. This network is flexible and can adapt to a wide range of input timescales. Further investigations could be realized to understand more in-depth the impact of the number of layers, especially on performance in more complex tasks, and on the characteristic dynamics in each layer. Device variability should also be investigated, with a specific focus on the impact of having variability in STNOs' relaxation times, which could help process series with multiple patterns at different timescales. Finally, the hardware realization of such a network could also be envisaged ; however, this requires physical synapses that are able to reach their steady state much faster than the neurons ; otherwise, their dynamical response needs to be taken into account.

List of publications and participations in conferences

COMMUNICATIONS

Oral Colloque Louis Néel, Sète *11.2023*

Fully Parallel Spintronic Convolutions

Oral JEMS conference, Madrid *09.2023*

Spin diode based Convolutional Layer with Frequency Connectivity

Oral MMM conference, Minneapolis *11.2022*

Fully-Parallel Convolution with Chains of Spin-Diodes

PUBLICATIONS

Article submitted to Physical Review Applied *2024*

Training a multilayer dynamical spintronic network with standard machine learning tools to perform time series classification

IEDM 2024 invited paper *2024*

Convolutions with Radio-Frequency Spin-Diodes

Résumé en français

6.12 . Introduction

De nos jours, l'intelligence artificielle (IA) est un domaine en pleine expansion. Ses applications sont nombreuses : applications médicales, conduite autonome, prédiction en temps réel des tendances économiques, et bien sûr, traitement du langage naturel. Le représentant le plus célèbre des modèles d'intelligence artificielle, le chatbot ChatGPT [1], est désormais présent dans la vie quotidienne de millions de personnes. Cependant, l'utilisation croissante de l'intelligence artificielle soulève une préoccupation majeure concernant sa consommation d'énergie. Les centres de données où les modèles d'IA sont hébergés et exécutés consomment environ 1 à 2% de la production mondiale d'électricité et cette consommation devrait doubler d'ici 2030 [2]. Cette consommation d'énergie doit être prise en compte afin de lutter contre le changement climatique.

Cette consommation d'énergie des centres de données est principalement causée par le décalage entre l'architecture matérielle traditionnelle et les exigences des modèles d'IA. Les matériels traditionnels sont basés sur une architecture où la mémoire et les unités de traitement sont séparées, cependant les réseaux neuronaux d'IA nécessitent l'accès à des milliards de paramètres en mémoire pour le traitement. Cela crée d'énormes fluxs de données entre la mémoire et le processeur, responsable de la majeure partie de la dissipation d'énergie. Pour éliminer cette cause de dissipation énergétique, une nouvelle approche pour le hardware spécialisé en IA est développée : le calcul neuromorphique. S'inspirant du cerveau, l'idée principale est de mettre en œuvre ces systèmes hardwares avec des dispositifs capables de fusionner les capacités de mémoire et de traitement. L'un des domaines qui fournit de tels composants est celui de la spintronique, où au lieu de manipuler des charges électriques, c'est le spin des électrons qui est l'élément principal pour coder et transmettre l'information. Ces composants sont capables de fonctionner avec une puissance inférieure à celle de l'électronique standard.

Dans cette thèse, je présenterai le travail que j'ai réalisé pour mettre en œuvre des réseaux neuronaux de différents types avec des composants spintroniques, en tirant parti des différentes propriétés offertes par ces composants. Ce travail se divise en deux parties : la réa-

lisation expérimentale d'un réseau neuronal convolutif (CNN) hybride matériel-logiciel, et la simulation d'un réseau multicouche de neurones dynamiques spintroniques.

Les composants spintroniques peuvent être adressés par des signaux radiofréquences. Cette connectivité en fréquence peut résoudre le problème majeur du câblage dans les réseaux neuronaux hardware. En effet, en mettant en œuvre un multiplexage en fréquence des signaux combiné avec des dispositifs sélectifs en fréquence, le nombre de connexions spatiales requises peut être considérablement réduit. Le réseau résultant présente une meilleure compacité et une géométrie plus simple. Dans la première partie de la thèse, nous démontrons la mise en œuvre contrôlée de ce multiplexage en fréquence à travers la réalisation d'une couche convolutive spintronique. Le chapitre 2 présente la réalisation de différents designs de synapses basées sur l'effet de diode à magnétorésistance anisotrope (AMR). Ces dispositifs appliquent un poids réglable sur les entrées radiofréquences. Ils sont constitués d'une simple couche magnétique métallique leur conférant robustesse et une fabrication relativement simple. Au chapitre 3, nous démontrons le concept d'opération de multiplication et d'accumulation avec ces synapses connectées en chaînes. Nous proposons une configuration mixte en série et en parallèle adaptée pour un futur passage à l'échelle. Nous démontrons un contrôle précis des poids synaptiques d'une chaîne de synapses fabriquée. Enfin, au chapitre 4, nous introduisons la mise en œuvre expérimentale d'un réseau neuronal convolutif avec une couche convolutive matérielle spintronique et une couche entièrement connectée logicielle. Ce réseau est entraîné en tenant compte du bruit provenant de l'installation expérimentale et correspond à la précision attendue sur une tâche de classification d'images de vêtements.

L'utilisation de la dynamique transitoire des dispositifs physiques peut offrir une manière économe en énergie de traiter les signaux variant dans le temps. Les nano-oscillateurs spintroniques (STNO) présentent des dynamiques non linéaires qui peuvent être exploitées pour de telles tâches. Dans la deuxième partie de cette thèse, je démontre qu'un réseau d'oscillateurs spintroniques couplés simulés avec une dynamique transitoire entraînable peut être utilisé pour classifier différentes séries temporelles. Au chapitre 5, je démontre qu'un réseau à une seule couche de ces oscillateurs peut être entraîné via la rétropropagation à travers le temps pour discriminer des séries temporelles sinusoïdales et carrées et montre que ses performances sont supé-

rieures à un réseau équivalent entraîné avec une approche de calcul par réservoir. Au chapitre 6, l'architecture d'un réseau multicouche de STNO est proposée et évaluée sur une tâche plus complexe, la discrimination de chiffres manuscrits injectés pixel par pixel. L'impact de la correspondance entre le temps de relaxation caractéristique des dispositifs et l'échelle de temps des entrées est analysé et des lignes directrices pour la réalisation tout réseau de neurones dynamiques sont proposées.

Chapitre 2

L'objectif du chapitre 2 est de démontrer la possibilité de réaliser des synapses spintroniques RF robustes et faciles à fabriquer. Ces synapses doivent être capables de traiter des entrées RF multiplexées en fréquence. Nous avons donc besoin de dispositifs sélectifs en fréquence et avec une réponse réglable, idéalement non-volatile. Nous avons choisi d'exploiter l'effet magnétorésistif anisotrope (AMR) pour réaliser de telles diodes. Cet effet convertit la puissance RF en une tension continue dont l'amplitude dépend de la fréquence d'entrée et de la fréquence de résonance du dispositif. Cette différence de fréquence représente le poids stocké par la synapse, et pour le modifier, la fréquence de résonance du dispositif est ajustée. Cet ajustement est réalisé ici en modifiant le champ magnétique externe. Nous avons proposé et étudié trois conceptions de synapses différentes : une basée sur une diode unique, une autre avec une diode plus une ligne de courant pour appliquer un champ d'Oersted local, et une dernière conception composée de deux diodes opposées avec un léger décalage de la fréquence de résonance. Ce décalage de la fréquence de résonance est implémenté par un décalage spatial entre les deux diodes qui se traduit par un décalage du champ magnétique externe appliqué. Cette dernière conception présente des avantages tels qu'un profil de fréquence net et un contrôle précis du poids synaptique. C'est donc le design choisi pour les développements suivants. Ce design, testé avec des diodes NiFe5/Pt5 de taille 5x10 μm, met en œuvre une synapse capable de traiter des entrées dans la gamme des GHz avec un poids non-volatile écrit en lithographie. Pour réaliser ces dispositifs, nous avons utilisé la lithographie UV. Un dispositif de caractérisation spécifique a été développé, intégrant une source pour les entrées RF, un nanovoltmètre et un ensemble d'aimants créant un gradient de champ magnétique contrôlé, permettant de contrôler le champ magnétique appliqué en déplaçant les dispositifs dans ce champ.

Chapitre 3

L'objectif du chapitre 3 est de fabriquer des chaînes de synapses adaptées à des opérations précises de multiplication et accumulation. Ces chaînes doivent comporter des synapses capables de traiter des entrées à différentes fréquences. Les synapses dans ces chaînes doivent avoir une réponse linéaire lorsqu'elles sont soumises à des entrées multiples. De plus, les poids implémentés à chaque fréquence d'entrée doivent être contrôlés avec précision, en tenant compte de la contribution de la synapse de redressement adressée, mais aussi des petites contributions des synapses aux fréquences voisines. Dans ce chapitre, nous présentons la vérification expérimentale de la linéarité des synapses fabriquées précédemment. Pour mettre en œuvre une dispersion de l'adresse de fréquence des synapses, celles-ci sont organisées en chaînes placées dans le gradient de champ démontré précédemment. Chaque diode est positionnée différemment le long de l'axe du gradient de champ, elle est donc soumise à un champ plus ou moins intense et possède sa propre fréquence de résonance. Des chaînes avec une configuration mixte parallèle-série pour les synapses ont été conçues afin d'obtenir une adaptation d'impédance à 50 Ohms. Pour obtenir le poids désiré dans une chaîne, nous avons conçu et mesuré une diode de référence utilisée comme modèle pour les diodes dans une chaîne. À partir de ce modèle de diode individuelle, nous pouvons modéliser une chaîne complète. Ce modèle est utilisé pour trouver la meilleure configuration spatiale des diodes afin d'obtenir des synapses avec les bonnes fréquences de résonance et les poids synaptiques corrects en utilisant des méthodes d'optimisation numérique. Avec ce design, nous avons fabriqué 3 chaînes comportant chacune 4 synapses. Les synapses sont espacées de distances d'environ 100 à 200 micromètres, correspondant à des décalages de fréquence de 5 GHz, et ont des décalages spatiaux internes d'environ ±50 μm pour implémenter les poids. Les poids obtenus après fabrication présentent une erreur moyenne de 5,2 $\%$.

Chapitre 4

L'objectif du chapitre 4 est de démontrer un réseau de neurones convolutionnel hybride matériel-logiciel avec une couche convolutionnelle spintronique, et de tester ce réseau sur une tâche complexe :

FashionMNIST[82]. Ce réseau est composé d'une couche convolutionnelle avec trois noyaux de 2x2, des neurones ReLU et une couche finale entièrement connectée. Ce réseau est d'abord entraîné purement en logiciel, cet entraînement inclut un bruit aléatoire appliqué sur la sortie de la couche convolutionnelle pour simuler le bruit expérimental, cette procédure aide le réseau à être résilient au bruit. Les poids optimaux trouvés pour la couche convolutionnelle sont ensuite extraits et trois chaînes de quatre synapses spintroniques sont fabriquées pour implémenter les douze poids des trois noyaux. La couche convolutionnelle nécessite donc que les images d'entrée soient envoyées sous forme d'entrées RF. Chaque image est décomposée en blocs de la taille des noyaux qui seront convertis en entrées RF et envoyés aux trois chaînes pour produire chacun un pixel de sortie. Cette opération est effectuée de manière séquentielle jusqu'à ce que l'image d'entrée entière ait été totalement analysée. Pour générer les entrées RF multiplexées en fréquence requises, nous avons fabriqué une source RF multicanaux personnalisée. Cette source est basée sur des oscillateurs à cristaux qui produisent un signal RF ajustable en fréquence, ces oscillateurs sont combinés avec des amplificateurs pour contrôler la puissance de sortie de chaque canal. Cette source a nécessité une calibration précise pour garantir que les entrées fournies sont correctes. La position de l'échantillon contenant les chaînes spintroniques est optimisée et corrigée au cours de la procédure d'évaluation pour compenser la dérive de sa position. Nous avons obtenu des résultats très satisfaisants, sur les 100 premières images du jeu de test, la précision expérimentale atteint 88%, comparable à la précision du logiciel avec bruit de 88,4% et légèrement inférieure à la précision du logiciel sans bruit de 90%.

Chapitre 5

Dans ce chapitre, nous démontrons la possibilité d'utiliser les dynamiques naturelles des STNOs pour traiter des entrées variant dans le temps. Nous proposons un réseau constitué d'une seule couche de STNOs couplés. Les dispositifs reçoivent des entrées sous forme de courants continus (DC) et leur sortie est la puissance RF émise. Ces courants continus injectés sont les paramètres sur lesquels sont appliqués les poids synaptiques. La dynamique des STNOs est simulée à l'aide d'une version simplifiée du modèle d'auto-oscillateur de Slavin et al.[77]. Le réseau ainsi obtenu est entraîné par rétropropagation du gradient tronquée à travers le temps pour classifier des séries temporelles composées d'ondes sinusoïdales et carrées. Cette tâche néces-

site à la fois de la non-linéarité et de la mémoire. Les performances de ce réseau entraîné sont comparées à celles de la même configuration traitée comme un système de calcul par réservoir, où seule la couche de sortie est optimisée et non les paramètres contrôlant la dynamique des neurones. Le réseau entraîné présente une précision plus élevée pour un même nombre de neurones : il peut atteindre une précision de 100% avec seulement deux neurones, tandis que l'approche par calcul par réservoir nécessite au moins 16 neurones pour obtenir le même résultat.

Chapitre 6

L'objectif de ce chapitre est de démontrer une architecture multicouche pour un réseau dynamique de STNOs sur une tâche complexe. L'autre objectif est d'examiner les contraintes liées à l'entraînement de ce type de réseau, en particulier en termes de correspondance entre l'échelle de temps des entrées et le temps de réponse caractéristique des dispositifs. L'architecture de réseau proposée est basée sur le réseau à une seule couche de STNOs couplés décrit précédemment. Cependant, les entrées de chaque couche doivent être centrées pour limiter la saturation des sorties des STNOs. Pour ce faire, des filtres passe-haut sont appliqués sur les sorties de chaque STNO. La tâche abordée est basée sur le jeu de données DIGIT [118] : des images manuscrites de chiffres 8x8 sont envoyées pixel par pixel au réseau, et l'étiquette correcte doit être prédite après avoir vu toute la série. Après une hyper-optimisation du taux d'apprentissage et du temps de relaxation des dispositifs à une échelle de temps d'entrée fixe, le réseau peut atteindre 89,83$\pm$2,91% de précision, à comparer aux 89,00$\pm$3,48% de précision atteints par un CTRNN standard avec le même nombre de neurones et une activation tanh. Nous avons également dérivé deux règles pour assurer un bon entraînement de ce réseau : le temps de relaxation des dispositifs doit être supérieur à l'échelle de temps des entrées pour garantir une mémoire suffisante dans le système, et le produit de la durée de l'entraînement, incluant les poids du réseau et l'échelle de temps des entrées, doit être proche de un. Cette deuxième règle limite la saturation des neurones, que ce soit par les bornes supérieures ou inférieures. Il est à noter que des paramètres tels que le taux d'apprentissage ou le temps de relaxation des STNOs peuvent varier dans une plage de cinq fois autour de leur valeur optimale sans dégrader la précision. Le réseau est capable de s'adapter à une large gamme de dynamiques. Un autre résultat important est que la densité

des connexions dans le réseau peut être réduite à 50% sans perte de précision, ce qui pourrait être un avantage pour les futurs développements hardware.

Bibliographie

[1] Standford university artificial intelligence index report 2024 (2024).

[2] Masanet, E., Shehabi, A., Lei, N., Smith, S. & Koomey, J. Recalibrating global data center energy-use estimates. *Science* **367**, 984–986 (2020).

[3] Nair, V. & Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814 (2010).

[4] Bottou, L. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91* (EC2, Nimes, France, 1991). URL http://leon.bottou.org/papers/bottou-91c.

[5] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *nature* **521**, 436–444 (2015).

[6] LeCun, Y. *et al.* Backpropagation applied to handwritten zip code recognition. *Neural computation* **1**, 541–551 (1989).

[7] Teo, Y. S. *et al.* Benchmarking quantum tomography completeness and fidelity with machine learning. *New Journal of Physics* **23**, 103021 (2021).

[8] Rangel, G., Cuevas-Tello, J. C., Nunez-Varela, J., Puente, C. & Silva Trujillo, A. A survey on convolutional neural networks and their performance limitations in image recognition tasks. *Journal of Sensors* **2024**, 1–29 (2024).

[9] Brown, T. B. Language models are few-shot learners. *arXiv preprint ArXiv :2005.14165* (2020).

[10] Turing, A. M. *Computing machinery and intelligence* (Springer, 2009).

[11] McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**, 115–133 (1943).

[12] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* **3**, 210–229 (1959).

[13] Christensen, D. V. *et al.* 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering* **2**, 022501 (2022). URL https://dx.doi.org/10.1088/2634-4386/ac4a83.

[14] Marković, D., Mizrahi, A., Querlioz, D. & Grollier, J. Physics for neuromorphic computing. *Nature Reviews Physics* **2**, 499–510 (2020).

[15] Amd virtex™ ultrascale+™ vu19p fpgas (2024). https://www.amd.com/fr/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus-vu19p.html [Accessed : (05/08/2024)].

[16] Soc adaptatif versal™ premium vp1902 (2024). https://www.amd.com/fr/products/adaptive-socs-and-fpgas/versal/premium-series/vp1902.html#product-brief [Accessed : (05/08/2024)].

[17] Abdelouahab, K., Pelcat, M., Serot, J. & Berry, F. Accelerating cnn inference on fpgas : A survey. *arXiv preprint arXiv :1806.01683* (2018).

[18] Mittal, S. A survey of fpga-based accelerators for convolutional neural networks. *Neural computing and applications* **32**, 1109–1139 (2020).

[19] Deng, J. *et al.* Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255 (Ieee, 2009).

[20] Wu, D. *et al.* A high-performance cnn processor based on fpga for mobilenets. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 136–143 (IEEE, 2019).

[21] Qiu, J. *et al.* Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, 26–35 (2016).

[22] Fpga et fgpa soc intel® stratix® 10 (2024). https://www.intel.fr/content/www/fr/fr/products/details/fpga/stratix/10.html [Accessed : (05/08/2024)].

[23] Geforce rtx 4090 (2024). https://www.nvidia.com/fr-fr/geforce/graphics-cards/40-series/rtx-4090/ [Accessed : (05/08/2024)].

[24] Wu, J. *et al.* Analog optical computing for artificial intelligence. *Engineering* **10**, 133–145 (2022).

[25] Zhang, Q., Yu, H., Barbiero, M., Wang, B. & Gu, M. Artificial neural networks enabled by nanophotonics. *Light : Science & Applications* **8**, 42 (2019).

[26] Zhang, T. *et al.* Efficient training and design of photonic neural network through neuroevolution. *Optics Express* **27**, 37150–37163 (2019).

[27] Tait, A. N. *et al.* Neuromorphic photonic networks using silicon photonic weight banks. *Scientific reports* **7**, 7430 (2017).

[28] Tait, A. N. *et al.* Microring weight banks. *IEEE Journal of Selected Topics in Quantum Electronics* **22**, 312–325 (2016).

[29] Mehrabian, A., Al-Kabani, Y., Sorger, V. J. & El-Ghazawi, T. Pcnna : A photonic convolutional neural network accelerator. In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, 169–173 (IEEE, 2018).

[30] Tait, A. N., Nahmias, M. A., Shastri, B. J. & Prucnal, P. R. Broadcast and weight : an integrated network for scalable photonic spike processing. *Journal of Lightwave Technology* **32**, 3427–3439 (2014).

[31] Ríos, C. *et al.* Integrated all-photonic non-volatile multi-level memory. *Nature photonics* **9**, 725–732 (2015).

[32] Cheng, Z., Ríos, C., Pernice, W. H., Wright, C. D. & Bhaskaran, H. On-chip photonic synapse. *Science advances* **3**, e1700160 (2017).

[33] Feldmann, J. *et al.* Parallel convolutional processing using an integrated photonic tensor core. *Nature* **589**, 52–58 (2021).

[34] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29**, 141–142 (2012).

[35] Caulfield, H. J. & Dolev, S. Why future supercomputing requires optics. *Nature Photonics* **4**, 261–263 (2010).

[36] Sun, Y., Dong, M., Yu, M., Liu, X. & Zhu, L. Review of diffractive deep neural networks. *Journal of the Optical Society of America B* **40**, 2951–2961 (2023).

[37] Chang, J., Sitzmann, V., Dun, X., Heidrich, W. & Wetzstein, G. Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification. *Scientific Reports* **8** (2018).

[38] Kuzum, D., Jeyasingh, R. G., Lee, B. & Wong, H.-S. P. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano letters* **12**, 2179–2186 (2012).

[39] Jagannadham, K. Microscopic mechanisms of filament growth in memristor. *Applied Physics A* **127**, 229 (2021).

[40] Chanthbouala, A. *et al.* A ferroelectric memristor. *Nature materials* **11**, 860–864 (2012).

[41] Wang, X., Chen, Y., Xi, H., Li, H. & Dimitrov, D. Spintronic memristor through spin-torque-induced magnetization motion. *IEEE electron device letters* **30**, 294–297 (2009).

[42] Chen, Y. *et al.* Polymer memristor for information storage and neuromorphic applications. *Materials Horizons* **1**, 489–506 (2014).

[43] Sun, Z. *et al.* Solving matrix equations in one step with cross-point resistive arrays. *Proceedings of the National Academy of Sciences* **116**, 4123–4128 (2019).

[44] Bayat, F. M. *et al.* Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nature communications* **9**, 2331 (2018).

[45] Wang, Z. *et al.* Fully memristive neural networks for pattern classification with unsupervised learning. *Nature Electronics* **1**, 137–145 (2018).

[46] Mehonic, A. *et al.* Memristors—from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing. *Advanced Intelligent Systems* **2**, 2000085 (2020).

[47] Pi, S. *et al.* Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nature nanotechnology* **14**, 35–39 (2019).

[48] Jackson, B. L. *et al.* Nanoscale electronic synapses using phase change devices. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* **9**, 1–20 (2013).

[49] Li, Y. & Ang, K.-W. Hardware implementation of neuromorphic computing using large-scale memristor crossbar arrays. *Advanced Intelligent Systems* **3**, 2000137 (2021).

[50] Fernando, B. R., Qi, Y., Yakopcic, C. & Taha, T. M. 3d memristor crossbar architecture for a multicore neuromorphic system. In *2020 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (IEEE, 2020).

[51] Ishii, M. *et al.* On-chip trainable 1.4 m 6t2r pcm synaptic array with 1.6 k stochastic lif neurons for spiking rbm. In *2019 IEEE International Electron Devices Meeting (IEDM)*, 14–2 (IEEE, 2019).

[52] Yao, P. *et al.* Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).

[53] Le Gallo, M. *et al.* A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nature Electronics* **6**, 680–693 (2023).

[54] Ambrogio, S. *et al.* An analog-ai chip for energy-efficient speech recognition and transcription. *Nature* **620**, 768–775 (2023).

[55] Wan, W. *et al.* A compute-in-memory chip based on resistive random-access memory. *Nature* **608**, 504–512 (2022).

[56] Krizhevsky, A., Hinton, G. *et al.* Learning multiple layers of features from tiny images (2009).

[57] Jinnai, B., Watanabe, K., Fukami, S. & Ohno, H. Scaling magnetic tunnel junction down to single-digit nanometers—challenges and prospects. *Applied physics letters* **116** (2020).

[58] Julliere, M. Tunneling between ferromagnetic films. *Physics letters A* **54**, 225–226 (1975).

[59] Jung, S. *et al.* A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature* **601**, 211–216 (2022).

[60] Borders, W. A. *et al.* Measurement-driven neural-network training for integrated magnetic tunnel junction arrays. *Physical Review Applied* **21**, 054028 (2024).

[61] Verma, G., Soni, S., Nisar, A. & Kaushik, B. K. Multi-bit mram based high performance neuromorphic accelerator for image classification. *Neuromorphic Computing and Engineering* **4**, 014008 (2024).

[62] Shibata, T. *et al.* Linear and symmetric conductance response of magnetic domain wall type spin-memristor for analog neuromorphic computing. *Applied Physics Express* **13**, 043004 (2020).

[63] Liu, L. *et al.* Domain wall magnetic tunnel junction-based artificial synapses and neurons for all-spin neuromorphic hardware. *Nature Communications* **15**, 4534 (2024).

[64] Lequeux, S. *et al.* A magnetic synapse : multilevel spin-torque memristor with perpendicular anisotropy. *Scientific reports* **6**, 31510 (2016).

[65] Yue, K., Liu, Y., Lake, R. K. & Parker, A. C. A brain-plausible neuromorphic on-the-fly learning system implemented with magnetic domain wall analog memristors. *Science advances* **5**, eaau8170 (2019).

[66] Bhowmik, D. *et al.* On-chip learning for domain wall synapse based fully connected neural network. *Journal of Magnetism and Magnetic Materials* **489**, 165434 (2019).

[67] Song, K. M. *et al.* Skyrmion-based artificial synapses for neuromorphic computing. *Nature Electronics* **3**, 148–155 (2020).

[68] Gomes, T. d. C. S. C. *et al.* Neuromorphic weighted sum with magnetic skyrmions. *arXiv preprint arXiv :2310.16909* (2023).

[69] Qiu, S., Zeng, J., Han, X. & Liu, J. On-chip skyrmion synapse regulated by oersted field. *AIP Advances* **14** (2024).

[70] Luo, S. *et al.* Voltage-controlled skyrmion memristor for energy-efficient synapse applications. *IEEE Electron Device Letters* **40**, 635–638 (2019).

[71] Fukami, S. & Ohno, H. Perspective : Spintronic synapse for artificial neural network. *Journal of Applied Physics* **124** (2018).

[72] Romera, M. *et al.* Vowel recognition with four coupled spin-torque nano-oscillators. *Nature* **563**, 230–234 (2018).

[73] Zahedinejad, M. *et al.* Two-dimensional mutually synchronized spin hall nano-oscillator arrays for neuromorphic computing. *Nature nanotechnology* **15**, 47–52 (2020).

[74] Zahedinejad, M. *et al.* Memristive control of mutual spin hall nano-oscillator synchronization for neuromorphic computing. *Nature materials* **21**, 81–87 (2022).

[75] Incorvia, J. A. C. *et al.* Capturing biological behavior in nanomagnetic artificial neurons and synapses for energy-efficient neuromorphic computing. In *Electrochemical Society Meeting Abstracts prime2020*, 31, 2040–2040 (The Electrochemical Society, Inc., 2020).

[76] Brigner, W. H. *et al.* Domain wall leaky integrate-and-fire neurons with shape-based configurable activation functions. *IEEE Transactions on Electron Devices* **69**, 2353–2359 (2022).

[77] Slavin, A. & Tiberkevich, V. Nonlinear Auto-Oscillator Theory of Microwave Generation by Spin-Polarized Current. *IEEE Transactions on Magnetics* **45**, 1875–1918 (2009). URL http://ieeexplore.ieee.org/document/4802339/.

[78] Deac, A. M. *et al.* Bias-driven high-power microwave emission from mgo-based tunnel magnetoresistance devices. *Nature Physics* **4**, 803–809 (2008).

[79] Tehrani, S. *et al.* Recent developments in magnetic tunnel junction mram. *IEEE Transactions on magnetics* **36**, 2752–2757 (2000).

[80] Ross, A. *et al.* Multilayer spintronic neural networks with radiofrequency connections. *Nature Nanotechnology* **18**, 1273–1280 (2023).

[81] Raimondo, E. *et al.* Reliability of neural networks based on spintronic neurons. *IEEE Magnetics Letters* **12**, 1–5 (2021).

[82] Xiao, H., Rasul, K. & Vollgraf, R. Fashion-mnist : a novel image dataset for benchmarking machine learning algorithms (2017). cs.LG/1708.07747.

[83] Leroux, N. *et al.* Radio-frequency multiply-and-accumulate operations with spintronic synapses. *Phys. Rev. Appl.* **15**, 034067 (2021). URL https://link.aps.org/doi/10.1103/PhysRevApplied.15.034067.

[84] Kittel, C. Ferromagnetic resonance. *J. phys. radium* **12**, 291–302 (1951).

[85] Tulapurkar, A. A. *et al.* Spin-torque diode effect in magnetic tunnel junctions. *Nature* **438**, 339–342 (2005). URL http://www.nature.com/articles/nature04207.

[86] Leroux, N. *et al.* Hardware realization of the multiply and accumulate operation on radio-frequency signals with magnetic tunnel junctions (2021).

[87] Leroux, N. *Artificial neural networks with radio-frequency spintronic nano-devices*. Theses, Université Paris-Saclay (2022). URL https://theses.hal.science/tel-03741830.

[88] Leroux, N. *et al.* Convolutional neural networks with radio-frequency spintronic nano-devices. *CoRR* **abs/2111.04961** (2021). URL https://arxiv.org/abs/2111.04961. 2111.04961.

[89] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **79**, 2554–2558 (1982).

[90] Jordan, M. Serial order : a parallel distributed processing approach. technical report, june 1985-march 1986. Tech. Rep., California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science (1986).

[91] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika* **71**, 6 (1986).

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

[93] Funahashi, K.-i. & Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks* **6**, 801–806 (1993).

[94] Cho, K. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *arXiv preprint arXiv :1406.1078* (2014).

[95] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).

[96] Vaswani, A. Attention is all you need. *arXiv preprint arXiv :1706.03762* (2017).

[97] Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany : German national research center for information technology gmd technical report* **148**, 13 (2001).

[98] Moore, E. H. On the reciprocal of the general algebraic matrix. *Bulletin of the american mathematical society* **26**, 294–295 (1920).

[99] Penrose, R. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, vol. 51, 406–413 (Cambridge University Press, 1955).

[100] Zhang, T. & Haider, M. R. A schmitt trigger based oscillatory neural network for reservoir computing. *Journal of electrical and electronic engineering* (2020).

[101] Chen, R. T., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems* **31** (2018).

[102] Pontryagin, L. S. *Mathematical theory of optimal processes* (Routledge, 2018).

[103] Hasani, R., Lechner, M., Amini, A., Rus, D. & Grosu, R. Liquid time-constant networks **35**, 7657–7666 (2021).

[104] Bueno, J. *et al.* Reinforcement learning in a large scale photonic recurrent neural network. *CoRR* **abs/1711.05133** (2017). URL http://arxiv.org/abs/1711.05133. 1711.05133.

[105] Li, Z. *et al.* E-RNN : design optimization for efficient recurrent neural networks in fpgas. *CoRR* **abs/1812.07106** (2018). URL http://arxiv.org/abs/1812.07106. 1812.07106.

[106] Long, Y., Jung, E. M., Kung, J. & Mukhopadhyay, S. Reram crossbar based recurrent neural network for human activity detection 939–946 (2016).

[107] Larger, L. *et al.* Photonic information processing beyond turing : an optoelectronic implementation of reservoir computing. *Optics express* **20**, 3241–3249 (2012).

[108] Dudas, J. *et al.* Quantum reservoir neural network implementation on coherently coupled quantum oscillators. *arXiv preprint arXiv :2209.03221* (2022).

[109] Pinna, D., Bourianoff, G. & Everschor-Sitte, K. Reservoir computing with random skyrmion textures. *Physical Review Applied* **14**, 054020 (2020).

[110] Gartside, J. C. *et al.* Reconfigurable training and reservoir computing in an artificial spin-vortex ice via spin-wave fingerprinting. *Nature Nanotechnology* **17**, 460–469 (2022).

[111] Nakane, R., Tanaka, G. & Hirose, A. Reservoir computing with spin waves excited in a garnet film. *IEEE access* **6**, 4462–4469 (2018).

[112] Marković, D. *et al.* Reservoir computing with the frequency, phase, and amplitude of spin-torque nano-oscillators. *Applied Physics Letters* **114** (2019).

[113] Torrejon, J. *et al.* Neuromorphic computing with nanoscale spintronic oscillators. *Nature* **547**, 428–431 (2017).

[114] Ababei, R. V. *et al.* Neuromorphic computation with a single magnetic domain wall. *Scientific Reports* **11**, 15587 (2021).

[115] Hughes, T. W., Williamson, I. A. D., Minkov, M. & Fan, S. Wave physics as an analog recurrent neural network. *Science Advances* **5** (2019).

[116] Qu, Y., Zhou, M., Khoram, E., Yu, N. & Yu, Z. Resonance for analog recurrent neural network. *ACS Photonics* **9**, 1647–1654 (2022).

[117] Rodrigues, D. *et al.* Dynamical neural network based on spin transfer nano-oscillators. *IEEE Transactions on Nanotechnology* **PP**, 1–6 (2023).

[118] Pedregosa, F. *et al.* Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[119] Furuhata, G., Niiyama, T. & Sunada, S. Physical deep learning based on optimal control of dynamical systems. *CoRR* **abs/2012.08761** (2020). URL https://arxiv.org/abs/2012.08761. 2012.08761.

[120] Azevedo, A., Vilela-Leão, L. H., Rodríguez-Suárez, R. L., Lacerda Santos, A. F. & Rezende, S. M. Spin pumping and anisotropic magnetoresistance voltages in magnetic bilayers : Theory and experiment. *Physical Review B* **83**, 144402 (2011). URL https://link.aps.org/doi/10.1103/PhysRevB.83.144402.

[121] Zeng, F. *et al.* Intrinsic Mechanism for Anisotropic Magnetoresistance and Experimental Confirmation in Co x Fe 1 x Single-Crystal Films. *Physical Review Letters* **125**, 097201 (2020). URL https://link.aps.org/doi/10.1103/PhysRevLett.125.097201.

[122] Dudas, J. *et al.* Quantum reservoir computing implementation on coherently coupled quantum oscillators. *Npj Quantum Inf.* **9** (2023).

[123] Paszke, A. *et al.* Pytorch : An imperative style, high-performance deep learning library. *CoRR* **abs/1912.01703** (2019). URL http://arxiv.org/abs/1912.01703. 1912.01703.

[124] Pascanu, R., Mikolov, T. & Bengio, Y. Understanding the exploding gradient problem. *CoRR* **abs/1211.5063** (2012). URL http://arxiv.org/abs/1211.5063. 1211.5063.

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

[126] Kingma, D. P. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980* (2014).

[127] Sutskever, I. *Training Recurrent Neural Networks.* Ph.D. thesis, University of Toronto, Canada (2013).