FPAA Project

Report

Bertrand THIA THIONG FAT & Yassine KAMRI

Metz, 2019

Table of content

Abstract - In this report, we will present the results of our work on the implementation of a brain-inspired computing system. The objective is to simulate a reservoir-computer that can be programmed on the DC RASP3.0A FPAA platform from Georgia Tech Institute, and thus, validate various experiences before reproducing them on the board in practice. At the end of the day, we will assess the performances of our simulator. We will finally validate its functioning and ability to simulate machine learning tasks. However, needless to say it is still a work in progress, and that further research needs to be done to improve our model.

1. Introduction

At the turn of the 21st century, with the tremendous progress of technology, Machine Learning has become a major field of interest to solve complex problems and build the world of tomorrow. It is indeed already used in security, medicine, and marketing. But this technology consumes a lot of energy, and the computations of such models can be long to train and construct.

With the development of new hardware and techniques, implementing machine learning algorithms on physical platforms is possible and more and more relevant. The analogic calculation is indeed thousands of time quicker, and inexpensive.

In this project, we will be working on the implementation of a reservoir computer in order to simulate machine learning tasks. It will allow the computations and validations of various settings easily, and thus make possible their implementation in a more effective way on an analogical hardware, such as the FPAA board in our disposition for example.

i. Objectives

The objectives of this project are:

- Develop a reservoir computer that can be used to perform Machine Learning tasks.
- Assess its performances on the example of digit spoken recognition.
- Evaluate the influences of the different adaptable parameters of the reservoir in its improvement.

We will be using Python in our work.

ii. Document structure

This document begins with an introduction to the concept of Reservoir Computing. We will then describe the notions behind this type of implementation of machine learning. After that, we will focus on a specific known task of classification: the digit spoken recognition one. This example will allow us to assess the performances of our model. Finally, we will try different settings to discuss the interest of the adaptable parameters regarding the potential improvement of our reservoir - such as the activation function or the connectivity of the reservoir.

2. A guide to understanding Reservoir Computing

This section will describe the key knowledge of the technology used in this project. We will present the structure of a reservoir computing and the way to train one. We will only focus on one model of reservoir, the one we used: The Echo States Network (ESN).

i. A field of interests

Recurrent Neural Networks represent a very powerful tool, integrating both large dynamical memory and highly adaptable computational capabilities. They are the Machine Learning model most closely resembling biological brains. However, their training is very difficult and demotivated many to exploit them.

Echo State Networks then appeared and became an alternative in computational neuroscience. The ESN approach allows indeed fast computations because of its intrinsic structure, and provides excellent performances in many known tasks. Another advantage is its potential to be implemented on non-conventional hardware, such as analog platforms for example – like our FPAA board.

ii. Architecture of an ESN model

The main power of the ESN resides in their simplicity - combining efficiency. Basically, the ESN are composed of a reservoir generated randomly, and only the readout from the reservoir is trained.

In a more practical way, their implementations can be defined as 3 categories of weights concatenated to each other: the input weights W_{in}, the internal weights W, and the output ones W_{out}.

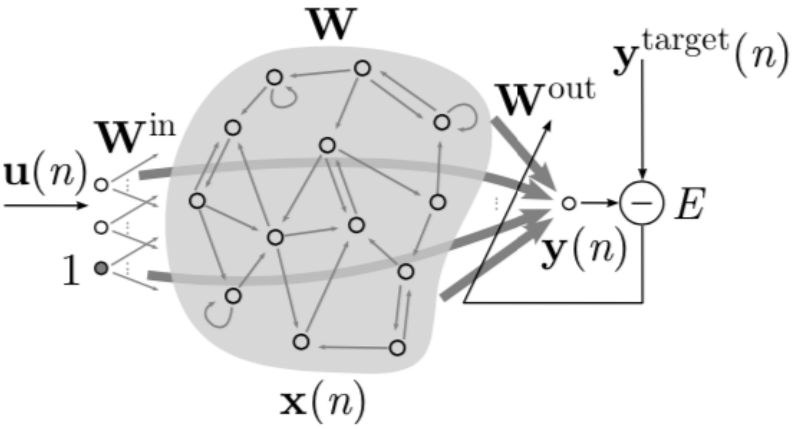


Figure 1 : An Echo State Network

As we described earlier, W_{in} and W are generated randomly, and only W_{out} has to be determined to have a functional and useful model.

On the previous scheme, $x(n)$ stands for the state of the reservoir, i.e. the values of the units, or nodes, of the ESN at the time n. The three weights hence represent the connectivity between those neurons, and the importance of the relations they have between each other. x depends on its previous values, and the ESN models are thus applied to supervised temporal ML tasks.

$u(n)$ is the input at the time n, and $y(n)$ the corresponding output, predicted by the reservoir. To train the ESN model and determine W_{out}, this output is compared to $y^{target}(n)$, the expected output desired. A measure of the distance between those two values is calculated and minimized to obtain the output weights. To compute the predicted values, the sole knowledge of W_{out}, u and x is needed.

We will go deeper into the details of each component of an ESN model in the next section Implementation. Overall, the equations describing the states of the reservoir and its output are quite simple, and fully explained after.

iii. Characteristics of the reservoir

For the ESN models, the connectivity of the reservoir is generally low (below 10% most of the time in the literature). The reason is to reduce the time complexity of the computation of the states vector x through time – by optimizing the matrix multiplications. We will hence take this into account in our further implementation. Apart from this, in general, sparsity of the reservoir does not affect the performance much and this parameter has a low priority to be optimized. It is mainly set to speedup computations.

The size of the reservoir should be at least equal to the estimate of independent real values the reservoir has to remember from the input to solve its task. Since this value is not always easy to determine, a good way to choose the size of the reservoir is to try smaller reservoirs, then scale to bigger ones until having satisfying results.

The non-zero elements of the reservoir are picked randomly. Usually, uniform distribution is popular for its continuity of values and boundedness, and has proved to perform well for many benchmarking tasks. So, that is the way we will implement our reservoir.

Finally, for the ESN approach to work, the reservoir should satisfy the so-called echo state property: the state of the reservoir $x(n)$ should be uniquely defined by the fading history of the input $u(n)$. It appears that there is no established rule to ensure this situation, but empirically. However, it has been observed that if the eigen value maximum of W is below 1, this property is present in most situations. So, we will pay attention to this value to realize our project.

3. Implementation

In this section, we will describe how we implemented the ESN model on Python. It is a practical guide, similar to an instruction manual. We explain here the concrete steps of the programming of the reservoir, which covers the essence of the theory behind it.

i. The useful Mathematical information

Here are all the basic information we need to implement our model on Python: the dimensions of the components of the reservoir, and the equations to compute the evolution of its states.

Notations (Even if we basically use the previous notations, we will recall them here to be clear and introduce new ones)

- $M(k, l)$ will refer to a matrix of dimension $k \ x \ l$.
- $n \in [1, T]$: discrete time and number of data points.
- N_x : total number of units in the reservoir.
- $u(n)$: input vector
- $y(n)$: output vector
- $x(n)$: states of the units vector
- W_{in} : input weights matrix
- W : internal weights matrix
- W_{out} : output weights matrix
- f will refer to a non-linear function

Dimensions

- $u(n) \in M(N_u, 1)$. The whole input u is then a matrix $M(N_u,\ T)$.
- $y(n) \in M(N_y, 1)$. The whole output y is then a matrix $M(N_y,\ T)$.
- $x(n) \in M(N_x, 1)$
- $W_{in} \in M(N_x, 1 + N_u)$
- $W \in M(N_x,\ N_x)$
- $W_{out} \in M(N_y, 1 + N_u +\ N_x)$

Equations

Evolution of the states of the units:

$$x(n+1) = f(W_{in} * \begin{pmatrix} 1 \\ u(n) \end{pmatrix} + W * x(n-1))$$

(We thus have $\begin{pmatrix} 1 \\ u(n) \end{pmatrix} \in M(1 + N_u, 1)$).

Computation of the predicted output:

$$y(n) = \ W_{out} * \begin{pmatrix} 1 \\ u(n) \\ x(n) \end{pmatrix}$$

(We thus have $\begin{pmatrix} 1 \\ u(n) \\ x(n) \end{pmatrix} \epsilon\, M(1 + N_u + N_x,\ 1)$).

ii. Training the reservoir

Method for temporal series predictions

As we mentioned previously, W_{out} is determined by minimizing a distance between the predicted output of the reservoir y and the expected value y^{target}. It exists many ways to evaluate this distance, and thus to train an ESN model. We chose to use the Mean-Square Error (MSE) for our distance, and the calculus of W_{out} thus becomes a regular least-square regression.

If we call X the following matrix:

$$X = \begin{pmatrix} 1 \\ u \\ x \end{pmatrix}$$

Then, we hence have the expression of the output matrix:

$$W_{out} = Y^{target} * X^T * (XX^T)^{-1}$$

And this expression can be adapted to many data $u^1, u^2, \dots, u^N$ and the expected output associated $y^1, y^2, \dots, y^N$, by concatenating horizontally the matrix $X^1, X^2, \dots, X^N$ and $Y^{target1}, Y^{target2}, \dots, Y^{targetN}$ as following:

$$X = (\, X^1 \quad \cdots \quad X^n)$$
$$Y^{target} = (\, Y^{target1} \quad \cdots \quad Y^{targetN})$$

This allows an even more powerful training.

However, the method above can only be used to have a temporal sequence for the output of the reservoir, and is therefore not adapted for a classification problem.

Method for classification predictions

For classification tasks, the output $y(n)$ of dimension $M(N_y, 1)$ must be set, so that $N_y = the\ number\ of\ classes$. Also, the expected target used for the training must be set so that $y^{target}(n)$ is equal to one in the dimension corresponding to the correct class and zero everywhere else. The model is then trained to approximate $y^{target}(n)$ for each single sequence $u(n)$.

Finally, the class of the whole input u is:

$$class(u) = \arg\max(\bar{y})$$

With $\bar{y} = \frac{1}{T}\sum_{k=1}^{T} y(k)$ and $y(k)$ the predicted output for the input $u(k)$.

So, the expression for the output matrix is:

$$W_{out} = Y^{target} * \bar{X}^T * (\bar{X}\bar{X}^T)^{-1}$$

With:

$$\bar{X} = \begin{pmatrix} 1 \\ \bar{u} \\ \bar{x} \end{pmatrix}$$

And as described previously, it is possible to perform more powerful training with several data by concatenating horizontally the right matrix, as followed:

$$\bar{X} = \left(\overline{X^1} \quad \cdots \quad \overline{X^n} \right)$$

This method is not the only way to train an ESN model for classification task, but works however very well in most cases. That is why we will be using it to compute the training of our reservoir in our project.

iii. Software implementation

We realized our implementation with Anaconda 3.5 environment, using Python 3.6.6 and the NumPy, Matplotlib and csv libraries.

The codes can be found by clicking on this link: https://gitlab-research.centralesupelec.fr/thiathiongfa_ber/fpaa_project

The dataset we used to train our ESN model for classification was preprocessed with Matlab and the toolbox AuditoryToolbox (easily accessible on the Internet).

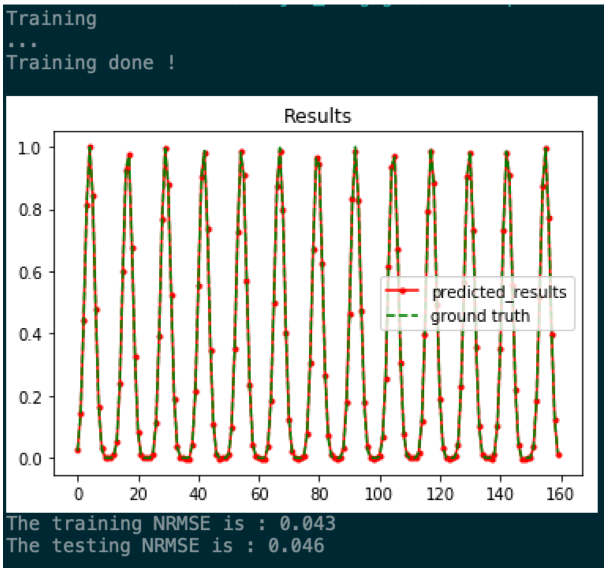
Finally, we tried to make the resumption of our work as effortless as possible, by commenting our lines when needed, and by describing the parameters and variables we instanced. Hopefully, this report will help to get an in-depth understanding of our work as well.

4. Testing of our implementation: obtaining a Sinus to the power

In this section, we decided to reproduce a known test to validate our implementation: obtaining a Sinus to the power. This task consists to generate the output $y(n) = \sin\left(\frac{n}{4}\right)^p$ from the input $u(n) = \sin\left(\frac{n}{4}\right)$, with p a positive integer value.

Here are the results for different settings (training on 240 data and testing on 160 data).

- For 100 units, a reservoir initialized randomly with a sparsity of 80% and computed so that the spectral radius of W is equal to 0.8.

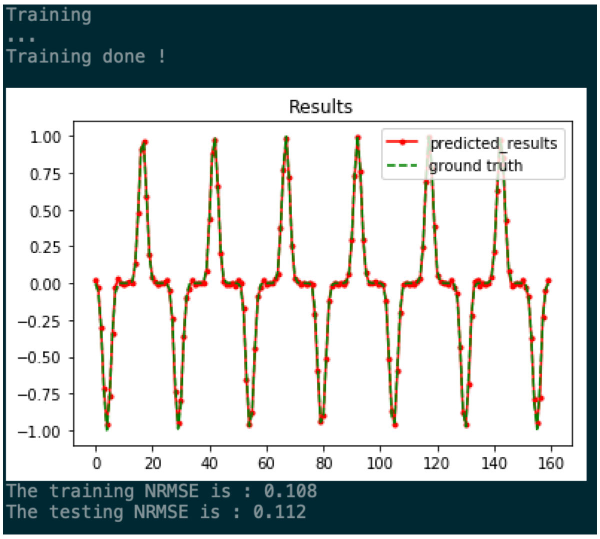


```
Training
...
Training done !
```

```
The training NRMSE is : 0.043
The testing NRMSE is : 0.046
```

Figure 2 : Obtaining sin^6

- For 100 units, a reservoir initialized randomly with a sparsity of 80% and computed so that the spectral radius of W is equal to 0.8.

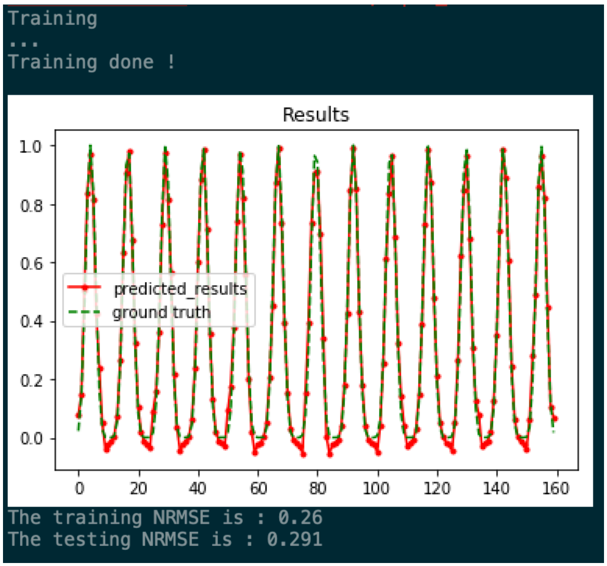


```
Training
...
Training done !
```

```
The training NRMSE is : 0.108
The testing NRMSE is : 0.112
```

Figure 3 : Obtaining sin^9

- For 20 units, and the same settings otherwise, we have:

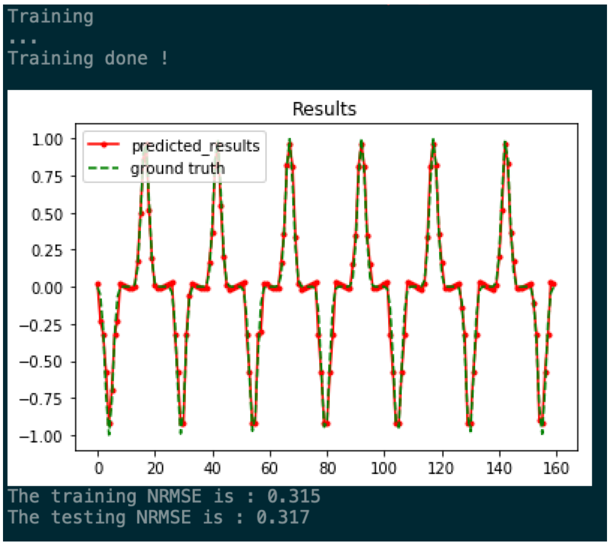


```
Training
...
Training done !
```

```
The training NRMSE is : 0.26
The testing NRMSE is : 0.291
```

Figure 4 : Obtaining sin^6

- Similarly, for an odd-numbered power:



```
Training
...
Training done !
```

```
The training NRMSE is : 0.315
The testing NRMSE is : 0.317
```

Figure 5 : Obtaining sin^9

We can observe that in both cases, we were able to generate the expected output, and this, with very satisfying results (low Normalized Root Mean Squared Error (NRMSE), even lower than the results from the work done by Son Dorian). Our model is consistent, even with various settings, and we can thus validate the efficiency of our implementation - indeed, we have the tendencies and evolutions expected while modifying the settings.

We chose here not to show more results because there were very similar to each other. Also, we will not further discuss about the influence of the settings of the reservoir on our results - such as the sparsity of W, its eigen value or the number of units - since this work has already been done by one of our classmates previously (San Dorian Nguyen).

The reason we computed this simple task was only to validate the implementation of our model in Python, before working on the classification task of Digit Spoken Recognition - which had never been done by any of our classmate before.

5. Presentation of the spoken digit recognition task

In this sub-section we present the classification task performed, as well as the dataset used and the pre-processing actions applied to the data.

i. Description of the task and Dataset

The second test used to characterize the performance of our reservoir computer corresponds to a classification task, more precisely the speech recognition of pronounced numbers. To carry out this test, we used the standard TI46 database provided by Texas Instrument. The TI46 corpus contains 16 speakers: 8 males labeled m1-m8 and 8 females

labeled f1-f8. There are 46 words per speaker. There are 26 utterances of each word from each speaker: 10 designated as training (or enrollment) tokens and 16 designated as testing tokens. The name of each file contains 8 characters followed by the .wav suffix. The first eight characters of the filename are formed by concatenating the 2-character prompt code

('00'-'YS'), the 2-character speaker code ('f1'-'f8','m1'-'m8'), the 2-character session code ('se' for enrollment, 's1'-'s8' for testing), and the 2-character token code ('t0'-'t9' for enrollment sessions, 't0'-'t1' for testing sessions). The organization of the corpus is as follows. The complete TI46 corpus is divided into two directories: TI20 and TI_ALPHA. TI20 contains all utterances of the words 'ZERO'-'NINE' and the words 'ENTER'-'YES'. TI_ALPHA contains all utterances of the words 'A'-'Z'. The TI20 and TI_ALPHA directories are each divided into TESTING and TRAINING directories, and each TESTING and TRAINING directory contains 16 sub-directories (one for each speaker). In each of the speaker directories, labeled F1-F8 and M1-M8, are the corresponding .wav files. All wav files begin with the standard 1024-byte NIST SPHERE format header, in which information pertaining to the file is stored. We did not succeed when we tried to read the files with python. Thus, we decided to switch to MATLAB for the preprocessing of the data.

ii. Passive Lyons ear model

This model is commonly used for speech recognition tests. It allows a particular information formatting, based on the functioning of the inner ear, which favors the proper conduct of the test. In addition, it is important to apply it during our tests so that they are comparable to the state of the art in the field that makes precisely use of this pre-treatment.

The human ear is divided into three parts. The outer ear that picks up acoustic vibration,

the middle and inner ear, which performs pre-processing of information.

The Lyon passive ear method used in our work is based on the biological functioning of the cochlea. It artificially reproduces the transformation of the acoustic wave happening in the middle and inner ear. This model consists of a battery of selective filters. The number of filters varies to adapt to the signals. As a result, this processing returns the cochleogram of the signal. The cochleogram is a matrix whose dimensions are the number of frequency channels (vertically) and the number of time samples required for the pre-processing (horizontally).

The cochleogram is composed of 78 frequency channels for all the recordings in our database. The number of time samples required for the total transformation of the original soundtrack for each recording is different because their length is different.

We represent here a cochleogram of a zero-digit record pronounced by a woman:

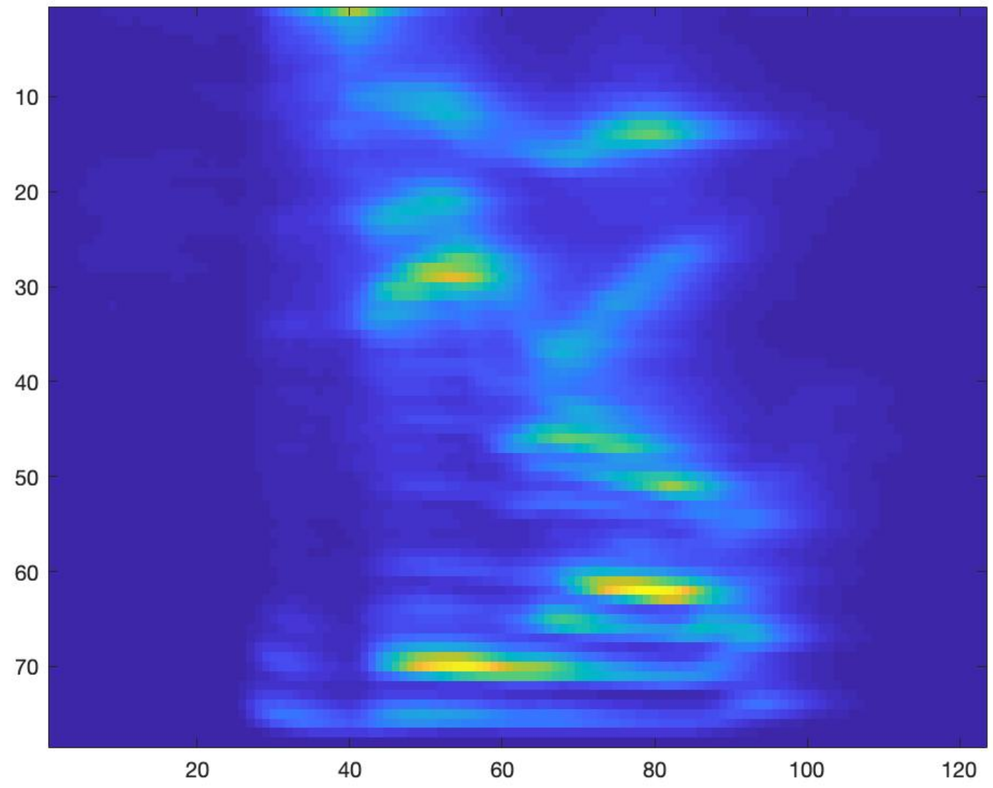


Figure 6 cochleogram of a zero-digit recording

To perform this pre-processing, we hence used a Matlab software toolbox, called the Auditory Toolbox, which is freely accessible at "http://cobweb.ecn.purdue.edu/malcolm/interval/1998-010/-010/".

iii. Second pre-processing layer depending on the architecture of our reservoir computer

In this sub-section, we describe a reservoir computing architecture compatible with an optical implementation as well as the necessary second pre-processing layer. We also discuss the differences with our architecture and how we adapt this second pre-processing to our architecture.

We studied an article entitled “Parallel photonic information processing at gigabyte per second data rates using transient states” that develops a certain model of all optical reservoir computing implementation. We are interested in optical computing methods because they represent a promising option to solve the problems of current computing methods. Indeed, the optical components could significantly increase the usable bandwidth and achieve efficient energy high performance computing. In this article, the researchers achieved this implementation by utilizing the analogue transient dynamics generated by a semiconductor laser coupled to a fibre-optic feedback loop. The form of the reservoir is as followed:

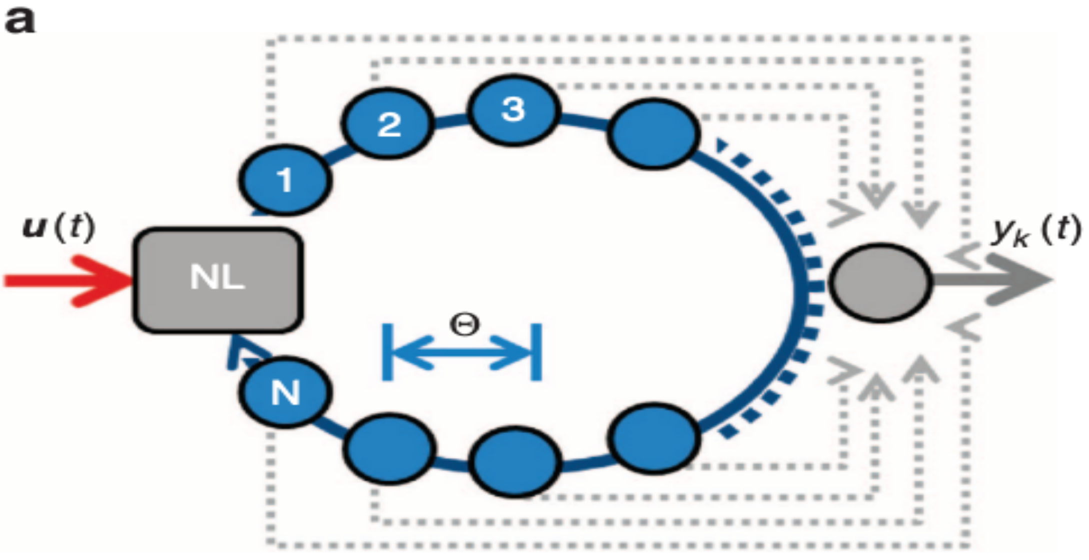


Figure 7 representation of non-linear time delayed reservoir computer

Schematic representation of computation using nonlinear transient states generated by a single nonlinear element (NL) subject to delayed feedback. The N transient states $x_m(t)$ used for computation are distributed along the delay line with spacing. Here, u stands for the information input, $y_k(t)$ for the value of the readout with index k.

An important part of the work we produced for our project is trying to reproduce the results obtained in this article by using simulation. To achieve this goal, we will present two types of reservoir computing system which are the non-linear delayed systems like the one implemented in this article and the neural network that we decided to simulate. It is important to make the difference between these two types in order to adapt correctly the pre-processing of the data following the implementation we choose.

We now recap some information about neural network-based reservoir computing system that we introduced earlier so we can explain the difference between the two types of reservoir computers:

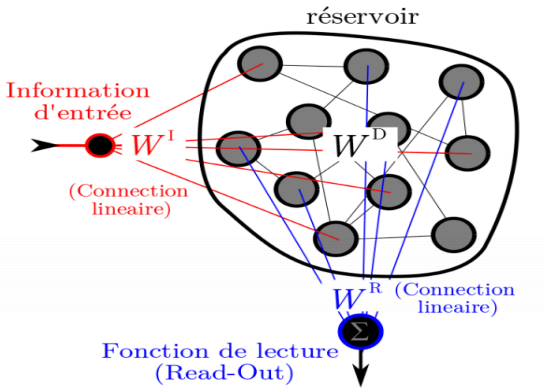


Figure 8 neural network-based reservoir computing

The spatial addressing of the neurons is performed using a W_I connectivity matrix. The values of the elements, fixed randomly, assign a weight to the connection between each neuron. The input information $u(n)$ is projected into a high-dimensional space, that of the reservoir or neural network. The reservoir connectivity matrix W_D, is set randomly and generally sparsely. The internal connections of the reservoir are defined by the W_D connectivity matrix and its elements determine the dynamics of the reservoir.

The state of the reservoir noted $x(n)$, is therefore a vector whose elements depend both on the input information $u(n)$ but also on the matrices W_I and W_D, as well as previous states of network nodes.

These features are to compare to those of a non-linear delayed reservoir computer.

We present the general principles, explaining how a non-linear delay physical system adapts to become a reservoir computing system. The main idea is to use a non-linear delay dynamic to replace the traditional reservoir defined as a spatially extended network.

The main argument on which this approach is based comes from the great complexity accessible with this physical system having a very large phase space, introduced by the temporal delay.

In this reservoir computing approach, the input information $u(n)$ is no longer injected simultaneously on all the nodes of the reservoir but must undergo pre-treatment in order to be serialized. To perform this function, we use a procedure that is illustrated as a time multiplexing, of $u(n)$, giving access to the addressing of the virtual nodes of the reservoir. The latter forming the spatial dimension are no longer physically accessible. As a result, we are now talking about virtual nodes. The spatial repartition of the neurons is replaced by a virtual temporal repartition.

The first time of the pre-processing procedure is to replace the W_D matrix by a matrix called "mask" M. This matrix represents a signal with a constant level on a time step of the system. This level is analog to the weight of a node in the WD matrix.

We represent such signal:

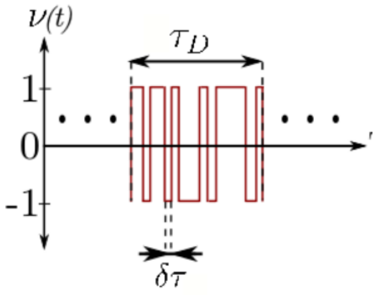


Figure 9 signal encoded by the mask

We multiply the input $u(n)$ by the mask M in order to emulate the spatial repartition of the nodes and the simultaneous repartition of the inputs on these nodes by a temporal repartition.

The internal connectivity of this reservoir computing approaches is controlled by several parameters. This is the separation of virtual nodes, time delay and system response time of the reservoir computer fixed by the low-pass or band-pass dynamics.

The article defines the mask used for the spoken digit recognition. In this matrix, 98% of the elements were set to zero in order to realize a sparse connectivity between cochleagram channels and transients. The remaining values were randomly selected among (0.41,0.59).

The results of this pre-processing are illustrated in the following figure. This connection is made by multiplying the cochleogram matrix with the mask.

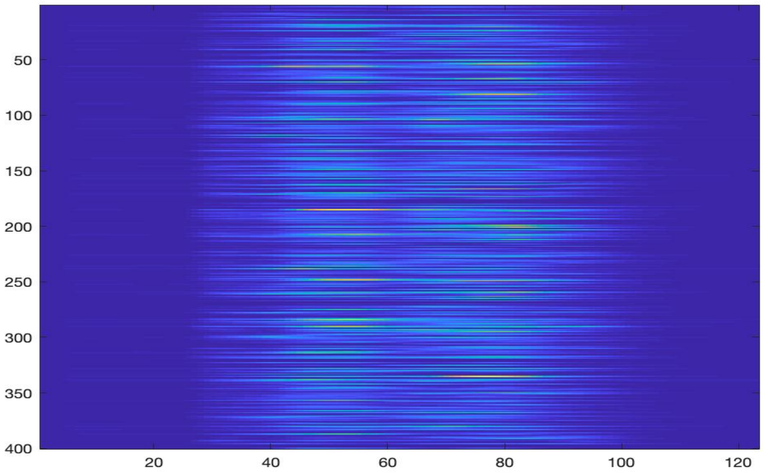


Figure 10 pre-processing results for a non-linear delayed system

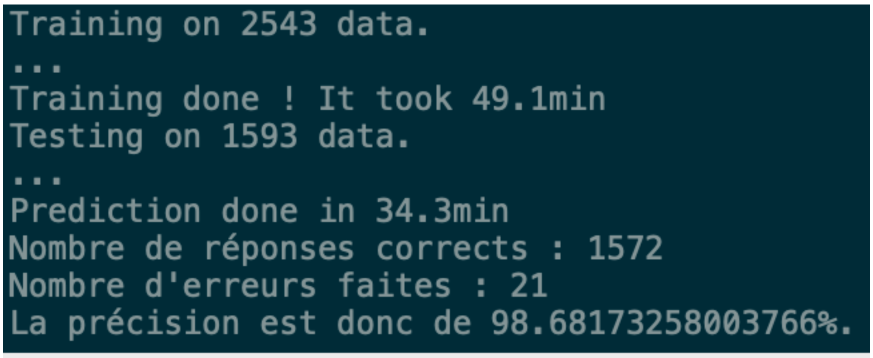
In order to adapt this pre-processing to our neural network-based implementation. We do not multiply the cochleogram by the mask. We instead choose the cochleogram as an input. To do so, we need to reshape the matrix into a simple vector by concatenating the columns of the matrix one after the other vertically. We also need to choose a connectivity matrix W_D of our neural network with the same form than the mask: 98% of the elements are set to zero and the remaining values were randomly selected among (0.41,0.59).

iv. Application: Training and testing our ESN model

We thus trained our model as described in the previous section with different settings. The best results we obtained were with the activation function sigmoid. The training was done with our whole dataset (the TI20 one), and the following parameters:

- sparsity: 0.98% of zero elements
- units = 400
- mask = [0.49, 0.51]
- activation = sigmoid
- dataset: TI20 (2543 data for the training set and 1593 for the testing set)

The results are shown below:

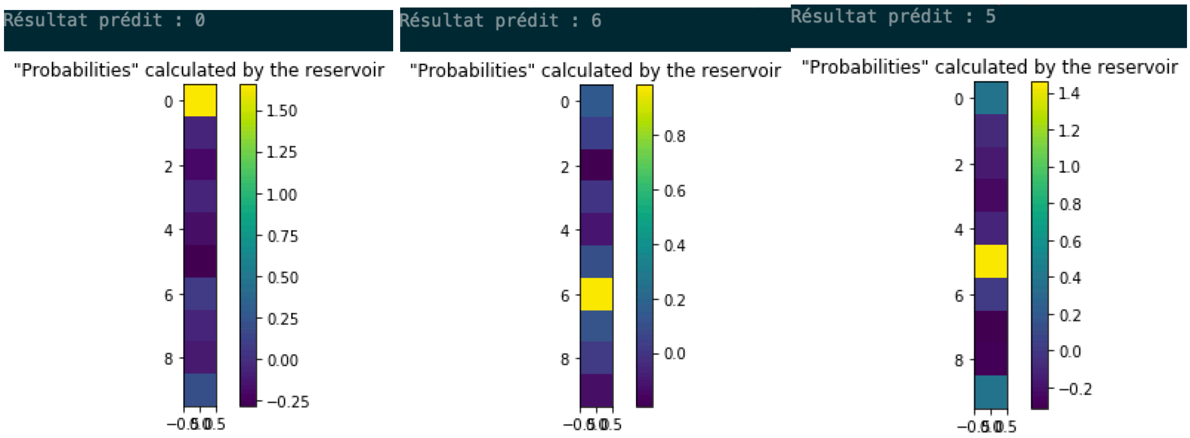


```
Training on 2543 data.
...
Training done ! It took 49.1min
Testing on 1593 data.
...
Prediction done in 34.3min
Nombre de réponses corrects : 1572
Nombre d'erreurs faites : 21
La précision est donc de 98.68173258003766%.
```

Figure 11 : Results of the "best" training

We obtained a success rate of 98.68%, which is very satisfying and close to the state-of-the-art. We can thus validate our implementation. Here are some results of our model on individual data from the testing set:

Prediction of a zero, a six, and a five:



```
Résultat prédit : 0
```

```
Résultat prédit : 6
```

```
Résultat prédit : 5
```

Figure 12 : Results of the training

As we can observe, the ESN model is capable of classifiying the different audio with a high level of confidence.

Now that our implementation is validated, let us change the settings of our reservoir to assess the influence of each of the adaptable parameters in its improvement.

6. Assessment of the settings regarding the improvement of our ESN model

We are going to assess the effects of several parameters on the performances of the reservoir on the spoken digit recognition task. First, we need to define a performance metric. We choose a very simple metric which is the word success rate defined by the ratio between the number of good identifications and the total number of inputs.

i. Influence of the amount of training data

We analyze the influence of the size of the training data on the performance of the reservoir. To do so, we performed multiple experiences with a fluctuating number of data, and we fixed the other parameters to the following values:

- Number of nodes: 400
- Probability of non-zeros values on the connectivity matrix W: 0.002
- Mask (connectivity matrix): as defined previously (but with a probability of zero values of 0.998)
- Activation function: hyperbolic tangent

The testing of the network is done on 500 randomly selected recording of the Texas Instrument database.

We precise that the mask was defined randomly only one time and used for every experiment. We explain the choice of a larger probability of zeros in the mask by the fact that the training of the network is faster which enabled us to run more experiments in a reasonable amount of time.

Theoretical expectations: Since the data are standardized by a reliable source (Texas Instrument) and the pre-processing was done correctly, we can consider that we have a dataset of good quality. The testing is also done on a dataset of the same quality. Therefore, we expect that the increase of training data will increase the performance of our system.

Experimental results:

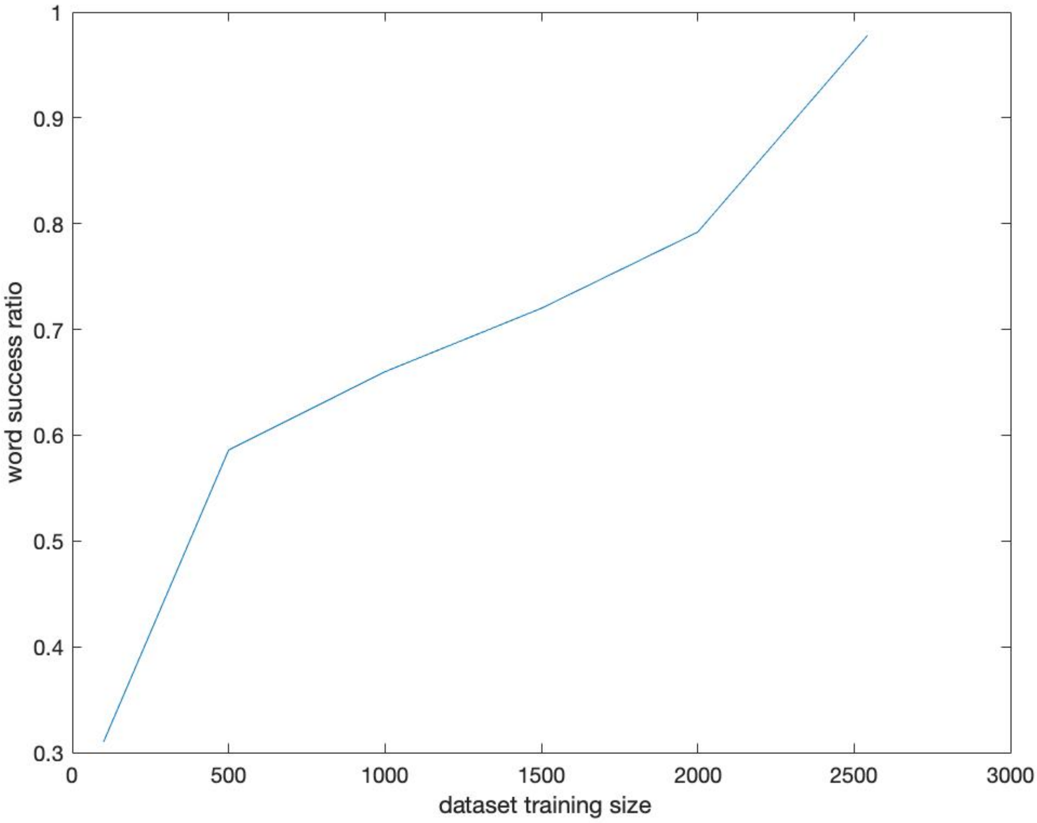


Figure 13 success rate for different amounts of training data

We remark that the tendency of this graph confirms our theoretical predeterminations although there are results fluctuations for a fixed number amount of data. We can explain these fluctuations by the random nature of the reservoir and by the random selection of testing inputs.

ii. Influence of sparsity on the performance

We analyze the influence of the sparsity on the performance of the reservoir. To do so, we performed multiple experiences with a fluctuating sparsity, and we fixed the other parameters to the following values:

- Number of nodes: 400
- Mask (connectivity matrix): as defined previously (but with a changing probability of zero values)
- Activation function: hyperbolic tangent
- Size of the training data: 500 recordings.

The testing of the network is done on 500 randomly selected recording of the Texas Instrument database.

Theoretical expectations: The connectivity of the reservoir weight matrix is considered to be the origin of the "richness" of the response signals in the reservoir, following this line of reasoning: sparse connectivity implies the decomposition of reservoir dynamics into loosely coupled subsystems which in turn leads to large fluctuations among the reservoir signals (desirable). However, many articles have reported that fully connected reservoirs work as well as sparsely connected ones. It appears plausible that a sparse random wiring does not lead to a dynamical decoupling, so the original intuitions are misguided. The relation between sparsity and performance is unclear. A more practically important aspect of a sparse connectivity is that it engenders linear scaling of computational complexity.

Experimental results:

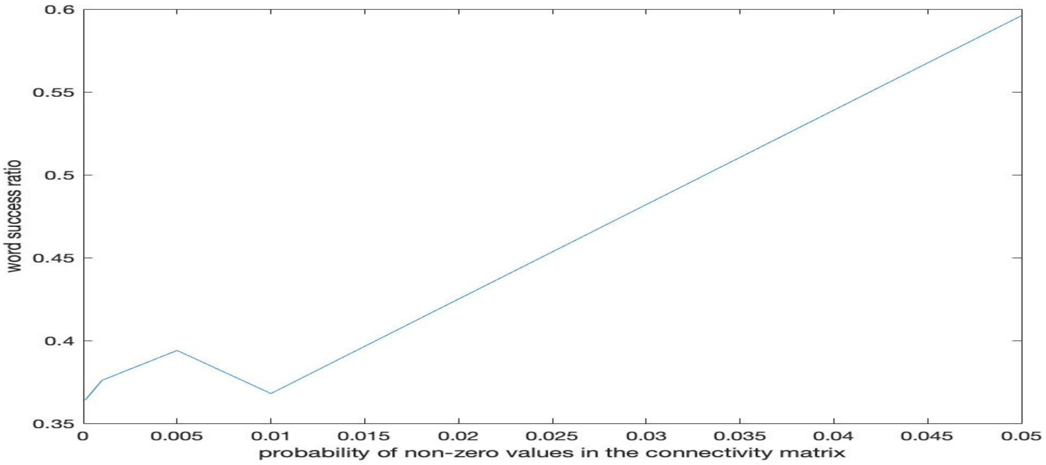


Figure 14 success rate for different sparsity coefficients

As theoretically expected, it seems difficult to draw a simple correlation between the sparsity of the connections and the reservoir performances.

iii. Mask non-zero values influence on performance

We analyze the influence of the mask values on the performance of the reservoir. To do so, we performed multiple experiences with a different mask values, and we fixed the other parameters to the following values:

- Number of nodes: 400
- Probability of non-zero values in the mask (WD matrix): 0.02
- Activation function: hyperbolic tangent

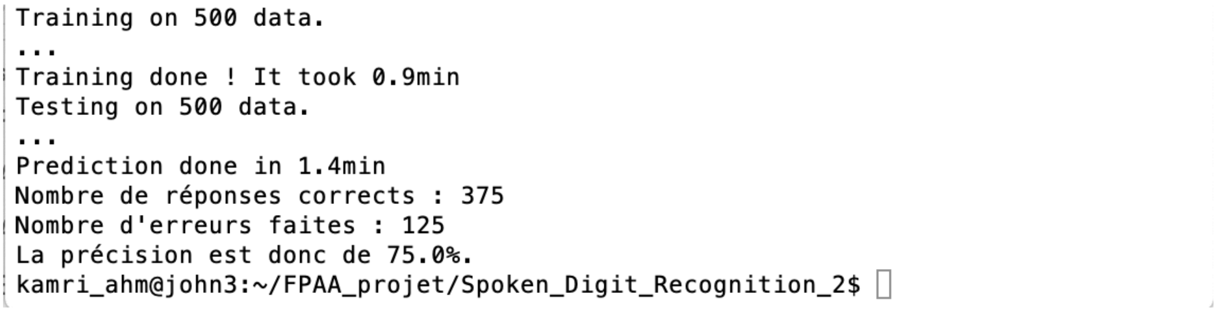
- Size of the training data: 500 recordings.

The testing of the network is done on 500 randomly selected recording of the Texas Instrument database.

Theoretical hypothesis: We empirically observed that the values of the mask must be positive and inferior or equal to one.

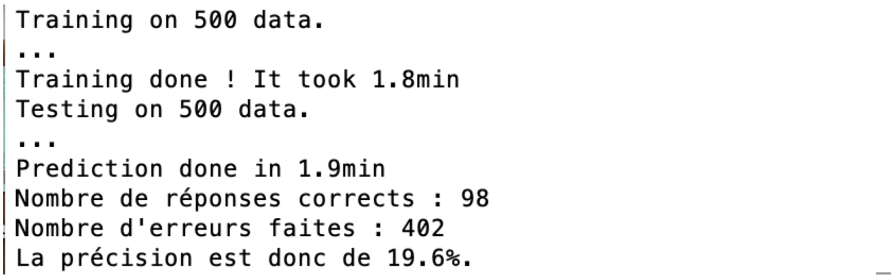
Experimental results supporting this claim:

We run experiments with different configuration of masks. We obtain our best results with a mask which coefficients are positives and inferior or equal to one. We give few examples of these experiments.



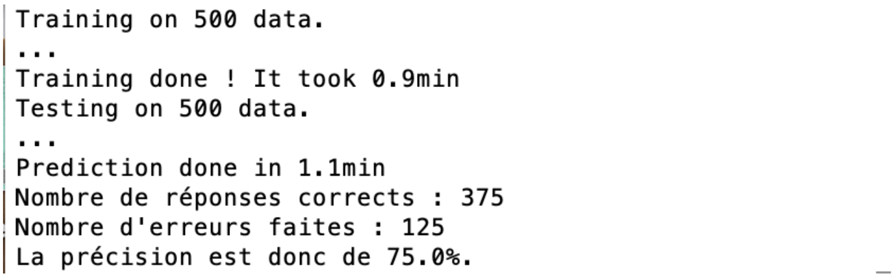
```
Training on 500 data.
...
Training done ! It took 0.9min
Testing on 500 data.
...
Prediction done in 1.4min
Nombre de réponses corrects : 375
Nombre d'erreurs faites : 125
La précision est donc de 75.0%.
kamri_ahm@john3:~/FPAA_projet/Spoken_Digit_Recognition_2$
```

Figure 15 results for the mask (0.49,0.51)



```
Training on 500 data.
...
Training done ! It took 1.8min
Testing on 500 data.
...
Prediction done in 1.9min
Nombre de réponses corrects : 98
Nombre d'erreurs faites : 402
La précision est donc de 19.6%.
```

Figure 16 results for the mask (1,-1)



```
Training on 500 data.
...
Training done ! It took 0.9min
Testing on 500 data.
...
Prediction done in 1.1min
Nombre de réponses corrects : 375
Nombre d'erreurs faites : 125
La précision est donc de 75.0%.
```

Figure 17 results for the mask (0.7,0.3)

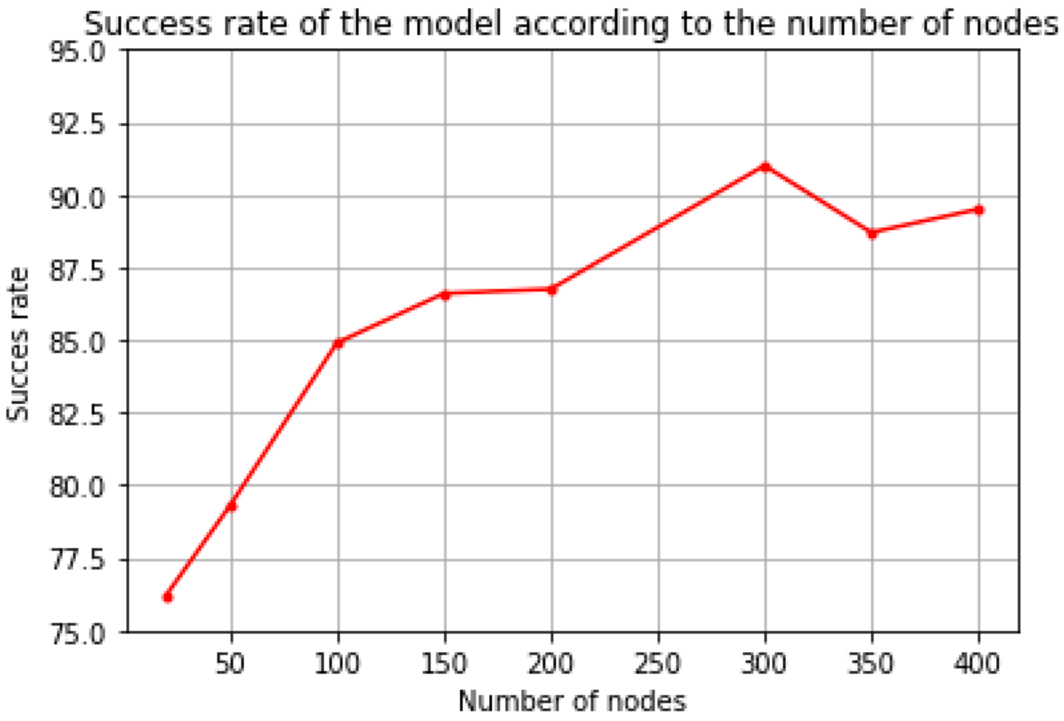
However, we are not able to fully understand the choice of the mask's coefficients due to the lack of information provided in the article about this choice and the lack of time to perform further research in the literature about the subject.

iv. Influence of the number of units

In this experience, we aimed at assessing the influence of the number of units on the performance of our model. Hence, we measured the success rate of our model after different trainings, and different sizes of reservoir, the other settings being fixed (2% of nonzero elements, mask of [0.49, 0.51] and the activation function sigmoid. We trained our model with a part of the dataset TI20, so the computation could be a bit faster (it will not be really faster actually, but if we took a smaller dataset, the results were not consistent).

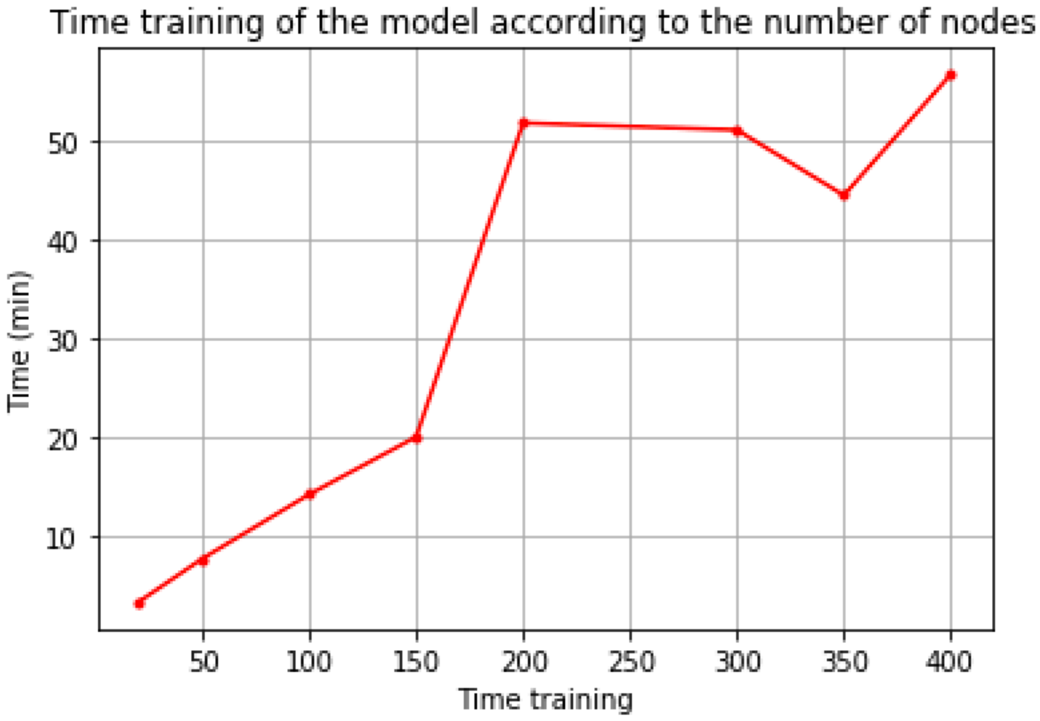
Theoretical expectations: The general expectation is that the bigger the reservoir is, the better the performances are. Indeed, the bigger the space of reservoir signals $x(n)$ is, and the easier it is to find a linear combination of the signals to approximate $y^{target}(n)$. However, above a certain number of units, we can also expect a lack of progress of the success rate, since the task would become too easy for the reservoir and it will not be capable of improving.

Experimental results:



We can observe that, as expected, the success rate increases when the number of nodes does the same. And the curb starts to stagnate. To make it clearer, it would have been better to perform a training with 500 nodes. However, we did not have the time.

As a matter of fact, the greater is the number of nodes, the greater lasts the time training, as we can see below:



And needless to say, that we need almost as much time to test the ESN model and assess its performances.

However, we can still read the graphs and conclude that our hypothesis was right: the bigger the number of nodes, the bigger the success rate. But at a certain point, the success rate seems not to increase anymore so it becomes useless to train the network. All the more that the chances of overfitting raise as well.

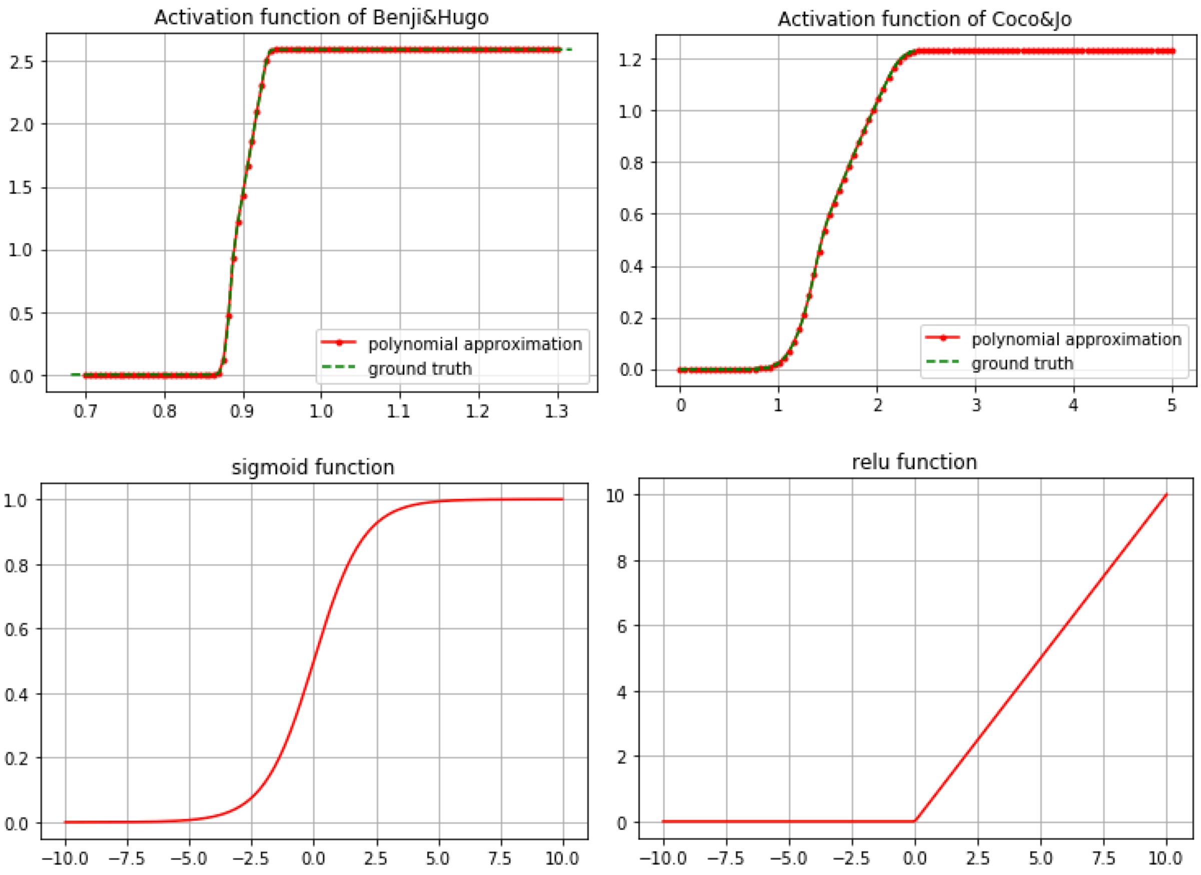
v. Influence of the activation function

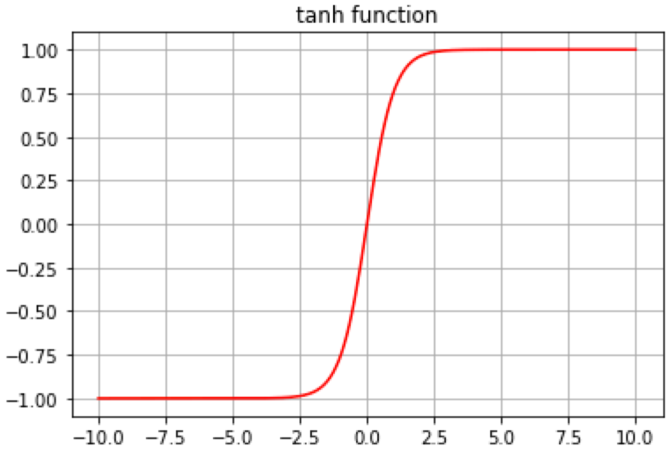
In this experience, we want to assess the influence of the choice of the activation function on the performance of our model. To do so, we fixed all the other settings and only changed the activation function, a training from another.

Here are the fixed parameters:

- sparsity = 0.98% of zero elements
- units = 400
- mask = [0.49, 0.51]
- dataset: TI20 (the whole dataset)

The activation function evaluated were:





The non-linearity functions from Benjamin&Hugo and Corentin&Joseph were obtained by approximating data points with high degree polynomials. Also, they are the results of neurons built by themselves, with the components present on the FPAA platform from Georgia Tech Lorraine. Our ESN model proves here its usefulness since it makes us possible to assess the potential of such neurons on the digit spoken recognition task.

Here is the overview table of the results:

Activation function	**Success rate**
Sigmoid	98.7%
tanh	98.55%
Non-linearity from Corentin&Joseph	95.8%
Non-linearity from Benjamin&Hugo	82%
relu	81.9%

It is difficult to analyze what makes a ‘good’ activation function for our classification task. Indeed, sigmoid and tanh both have the highest success rate and are antisymmetric. However, the non-linearities from Benjamin&Hugo and Corentin&Joseph are very similar but performs quite differently. The great success rate might be due to the saturation level around 1 of those three functions, but we did not have the time to investigate deeper and evaluate functions alike. Still, our simulator allows us to observe that the neurons created by Corentin&Joseph has the potential to perform very well on our classification task. It can therefore be interesting to reproduce the settings of our simulation in practice, on the FPAA platform, to validate this result.

General conclusion

We were able to implement a reservoir computing system that provided satisfactory results for the spoken digit recognition task to a certain extend. However, we still identified several leads to improve our work:

- Use noisy data to improve the robustness of the implementation.
- Explore other data pre-processing methods.
- Explore other training methods to avoid issues like overfitting (for instance Ridge regression)
- Search for the best activation function
- Understand what defines a good choice of coefficients for the mask and choose realistic ones that can be created with the FPAA board.
- Try to understand the influence of sparsity and how to choose a coherent one.
- Test the influence of other parameters like the Input Scaling and Leaking rate.