
UNIVERSITAT DE BARCELONA

**Treball de Fi de Grau**

**GRAU D'ENGINYERIA INFORMÀTICA**

**Facultat de Matemàtiques i Informàtica**
**Universitat de Barcelona**

---

# REVEALING BRAIN STATES WITH RESERVOIR COMPUTING

---

**Arnau Naval Ruiz**

Director: Ignasi Cos
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 28 de Novembre de 2020

# Contents

# List of figures

# List of tables

# 1. Introduction

Research in Computational Neuroscience faces immense challenges when it comes to explaining brain function. Not only are structures in the brain too complex to be entirely understood and represented, but collecting the right empirical data and devising the right computational approach to best characterize function are significant challenges as well. Here we propose to study the use of Reservoir Computing (RC) to address some of these issues.

RC is a relatively novel technique with of potential in several fields, including Computational Neuroscience. It is also an increasingly popular technique for neural data processing, as it shows promise to faithfully *represent temporal or sequential datasets*. As seen in Mattia Rigotti et al., (2010). It was proposed that rather than prewiring a network for the relevant representations for a task, a randomly connected network would be able to represent all configurations of relevant task stimuli. That is where RC comes into play, as the Reservoir model is built with these random connections.

Pierre Enel et al., (2016) provided an example application of Reservoir Computing to capture the neural dynamics in the prefrontal cortex, in the context of a problem-solving task performed by a primate. By the end of the training, the model was able to reproduce the task almost perfectly, so it was of interest to compare the neural encoding of the model to that of the primate. Although they conclude that the cortex is most likely encoded with more complicated schemes, it is still a significant finding made possible by RC.

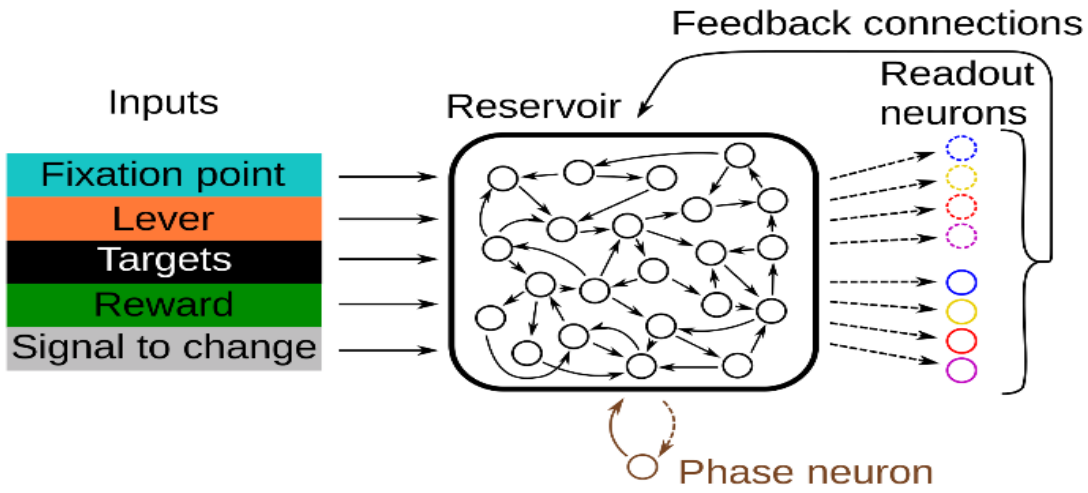



*Figure 1: RC Model used by Pierre Enel, 2016*

Figure 1 shows the structure of an RC model, consisting of an input layer, a reservoir layer, and an output layer. In this case, the model has an extra neuron called a “Phase neuron”, which encodes the current phase of the model (either search or repetition). we will not use a phase neuron like this in our tests.

In brief, here we propose to study the use of RC to classify movement-related brain states based on neuro-physiological data. The summary is described next.

## 1.1. Project Summary

The purpose of the project is to build a Reservoir Computing model capable of identifying different movement related states from a neural data set recorded from a non-human primate. To this end, we will perform the following analyses:

- Five-State Classification: where we will try to distinguish all five states at the same time, giving it a full trial to classify.

- Left vs Right: where we will try to distinguish when the data is from the primate using the left arm or the right arm.

Not only it is interesting to see if RC is capable of capturing the features recorded from specific brain regions, we will also see if the representation of the data after going through the model helps classify the states into their belonging classes better than without using the neural network.

All the code can be found in my GitHub repository:
https://github.com/ArnauNaval/TFG_ReservoirComputing

## 1.2 Previous work

Our project builds on previous analyses of the same dataset performed by Michael DePass' MSc Thesis (2019), aimed at characterizing neural states of movement execution before and after traumatic intervention. He used three feature types for classification purposes: spectral power, and functional connectivity which consisted of inter-signal covariance and correlation. He utilized two different classifiers: Multinomial Logistic Regression (MLR) and 1-Nearest Neighbor (1-NN). These classifiers were independently fit to each of the three types of metrics for data from each frequency band.

After testing, Michael found that MLR outperformed 1-NN and achieved Area Under the Curve (AUC) scores of above 0.8 for all tasks when fit to the highest frequency band power data and that lower frequency bands yielded considerably worse accuracies.

With our work we aim to reproduce these results and as mentioned in section 1.1., we will also try to improve these classifications by using Reservoir Computing.

# 2. State of the Art: Recurrent Neural Networks and Reservoir Computing

## 2.1. Recurrent Neural Networks

Recurrent Neural Networks (RNN) are commonly used when dealing with sequential data, as they retain memory of past events due to their recurrent structure. In other words, an RNN performs the same function for every input of data in addition to the recurrent input, which depends on past computation. After producing the output, it is copied and fed back into the recurrent network. In order to make a decision, it considers the current input and the output that it has learned from the previous input.

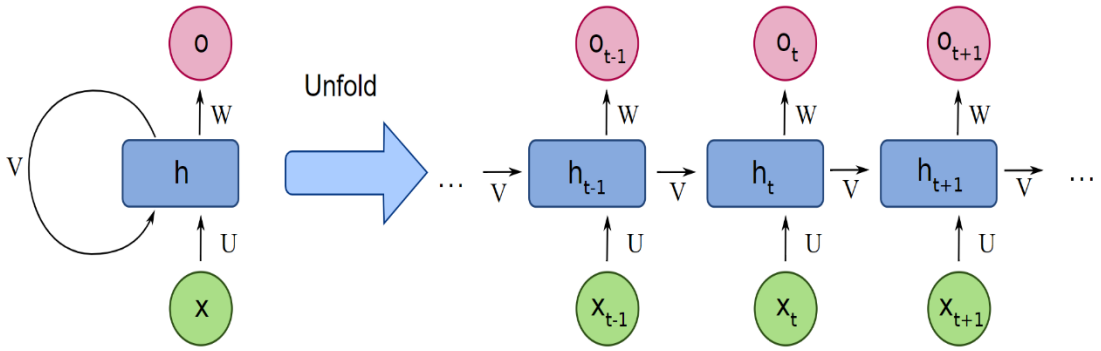



*Figure 2: Image of a vanilla RNN*

where $h_t = tanh(V\ h_{t-1} + U\ X_t)$ are the hidden states and $o_t = softmax(W\ X_t)$ are output states at each timestep *t*.

*Reservoir Computing (RC)* (Maass et al., 2002; Jaeger et al., 2001) is a derivation of an RNN, where the inputs are mapped into higher dimensional computational spaces through the dynamics of a fixed, non-linear system called a reservoir. It is a highly efficient and bio-inspired approach for processing time dependent data. RC involves a network of generic units with random connections in order to reproduce the temporal dynamics of the neural data.

Once the input is fed into the reservoir, which stores information using recurrent loops randomly organized around its nodes, a readout mechanism is trained to read the state of the reservoir and map it to the desired output. The key benefit of this model is that training only occurs at the readout stage, as the dynamics of the reservoir are fixed, thus reducing the computational complexity of the algorithm.

A type of RC that uses a large sparsely connected hidden layer (reservoir layer) where the connection weights are fixed and randomly generated is called *Echo State Network (ESN)*. The main interest of this network is that although it exhibits a non-linear behavior, it is possible to use a linear combination of the output layer in order to train its weights. We will see more about it in the following sections.

## 2.2. Reservoir Computing

Reservoir computing (RC) is a learning technique used to deduce the underlying dynamics given a set of sequential data points. For instance, it may be able to predict future data by learning its dynamics or it may produce a related output sequence by learning the dynamics of the input sequence. The neural network is composed by a single hidden layer along with an input and output layers.

Reservoir computing is a recurrent neural network approach where all connections within the network are fixed except for the output ones, which are the only connections to be learnt. All connections except the output will remain unchanged at all times. As the output connections are the only ones to be adjusted, Reservoir Computing requires a lower number of parameters to be trained, making it computationally cheaper than other RNN approaches. This is thanks to its sparse structure and to the so-called echo states, which is the key concept behind reservoir computing.

### 2.2.1. Structure of the neural network

The structure of the RC network we are going to be working with can be divided as follows:

- K-dimensional input denoted by $u$.
- N-dimensional hidden layer denoted by $x$.
- L-dimensional output denoted by $y$.

The values of the different elements in the network will be updated at each time step n, which will define the dynamics of the network.

Each of the elements can be defined as a multidimensional time series:

- $u(n)^T = (u_1(n), u_2(n), ..., u_K(n)))$ is a K-dimensional vector giving the input units at time n.
- $x(n)^T = (x_1(n), x_2(n), ..., x_N(n)))$ is an N-dimensional vector giving the internal units at time n.
- $y(n)^T = (y_1(n), y_2(n), ..., y_L(n)))$ is an L-dimensional vector giving the output units at time n.

In order to determine the connections between the layers we have the connection matrices:

- Input connections, $W^{in} \in M(\mathbb{R})_{N\times K}$, matrix with the connection weights from the input units to the internal units. $W^{in}_{i,j}$ gives the weight of the connection from the $j$-th input unit to the $i$-th internal unit.

- Internal connections, $W \in M(\mathbb{R})_{N\times N}$, matrix with the connection weights between the different internal units. $W_{i,j}$ gives the weight of the connection from the j-th internal unit to the i-th internal unit.

- Feedback connections, $W^{back} \in M(\mathbb{R})_{N\times L}$, matrix with the connection weights from the output units to the internal units. $W^{back}_{i,j}$ gives the weight of the connection from the j-th output unit to the i-th internal unit. In our case we have not used feedback connections as they didn't improve the accuracy on the tests.

- Output connections, $W^{out} \in M(R)_{L\times N}$, matrix with the connection weights from the internal units to the output units. $W^{out}_{i,j}$ gives the connection weight from the j-th internal unit to the i-th output unit.

Once the structure of the network is determined, we can introduce how the dynamics manifest in the network.

Internal states *X* are going to be updated at each timestep *n* as follows:

$$\mathbf{x}(n+1)=f(\mathbf{W}\mathbf{x}(n)+\mathbf{W}_{in}\mathbf{u}(n+1)+\mathbf{W}_{back}\mathbf{y}(n))$$

where $f$ is the activation function.

It can bee seen from the equation that the hidden state is computed from the past one. The hidden State, therefore, is an echo of the past. This allows us to deterministically define the connection matrices $\mathbf{W}_{in}$ and $\mathbf{W}$, in such a way that the only weights to be learned are those associated with the output ($\mathbf{W}_{out}$).

On the other hand, the equation that computes the output *y* is as follows:

$$\mathbf{y}(n+1)=f^{out}(\mathbf{W}_{\text{out}}\mathbf{x}(n+1))$$

where $f^{out}$ is the activation function.

The most commonly used activation function in RNNs is the hyperbolic tangent (*tanh),* which we also used in the tests that we performed.

### 2.2.2. Echo State Network

An Echo State Network (ESN) is an instance of the more general concept of Reservoir Computing. The basic idea behind the ESN is to benefit from the RNN structure, but without the training problems of a traditional RNN, such as vanishing/exploding gradient problems --- see sections next.

This is achieved with a relatively large reservoir of sparsely connected neurons. These connections are randomly assigned at the beginning and remain unchanged. In other words, the reservoir weights are not trained. Like reservoir connections, input connections are also randomly assigned upon initialization, and its weights are not trained. Input neurons feed the activations into the reservoir. On the other hand, the

output layer is fully connected to the reservoir, and its weights are the only ones that will be trained.

In training, once all the inputs have been fed into the reservoir, a simple application of linear regression to fit the relationship between the captured reservoir states and the target output targets.

The main idea is that the some of the sparse random connections in the reservoir will create loops between the neurons, allowing previous states to "echo" even after they have passed, so in case the network receives a novel input that is similar to one it has been trained with, the dynamics of the reservoir will start to follow the appropriated activation trajectory for this input, so it can provide a matching signal to the one it was trained with.

One of the main advantages of ESN is that only the output weights need to be trained, while still being able to capture complex temporal dynamics and to model properties of dynamical systems.

An important notion to keep in mind when working with ESN is that the **W** matrix needs to have a spectral radius $|\lambda_{max}| < 1$, where $\lambda_{max}$ is the largest eigenvalue of **W** in module. Then, the network will not have any asymptotically unstable null states.

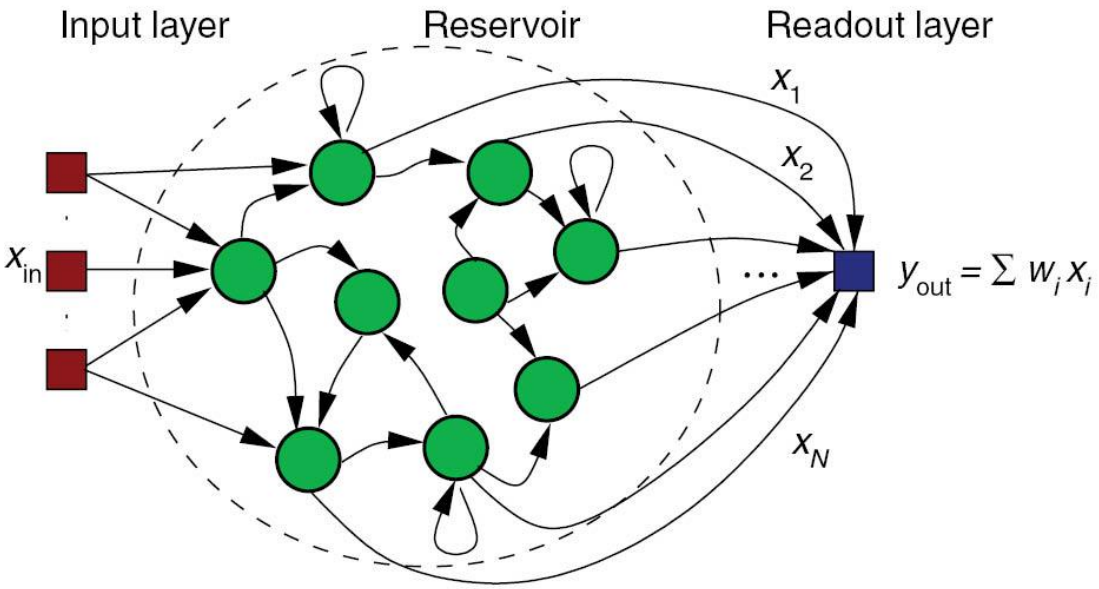



*Figure 3: Image of an Echo State Network*

Figure 3 shows that the connections between the input layer and the reservoir as well as the connections between the reservoir units are sparse (as opposed to fully connected). Also, some of the reservoir nodes have recurrent loops that will act as a short-term memory. By contrast, the reservoir layer is fully connected to the output layer.

### 2.2.3. RNN limitations

The Vanishing/Exploding gradient problem is found in certain artificial neural networks with gradient descent-based training methods, like *backpropagation.* This problem makes it difficult to learn and fine-tune the parameters of earlier layers in the network, and it gets worse as the number of layers is increased.

In a network with N hidden layers, N derivatives will be multiplied with each other, if these derivatives are large, the gradient will increase exponentially as we backpropagate down the model until it eventually explodes, so that a small change in the input, would result in massive changes in the weights of the model, making it extremely difficult to train. On the other hand, if the derivatives are small, the gradient will decrease exponentially as we backpropagate down the model until it eventually vanishes, so even a big change in the input, would result in very small changes in the weights of the model, also making it difficult to train.

In the case of ESN, as we use a linear function to compute the weights, we do not require the use of back propagation to train them, thus avoiding both of these problems.

### 2.2.4. Input/State forgetting property

Echo State Networks satisfy the input forgetting property, that is, the initial value of the input will only influence the dynamics of the nodes on the first time steps and, after some time, the trajectories will converge to the same one.

This is useful as it means that it does not matter which are the first inputs to the network, as the trajectories will converge to the same one, thus showing results not influenced by the initial values.

Also, they satisfy the state forgetting property, that is, that is, the initial state of the reservoir will only influence the dynamics of the nodes on the first time steps and, after some time, the trajectories will converge to the same one.

This is useful as it means that we can initialize the state matrices as any random number, but that will not influence the future results.

# 3. Materials & Methods

## 3.1. Neural Data

In order to train this model, we will be using a *Local Field Potential* (LFP) dataset collected by the Dancause Laboratory at the University of Montreal.

Electrophysiological neural data has been used for more than a century, with the first known recordings taking place 145 years ago Caton et al., (1875). This data modality constitutes an increasingly important tool both in neurophysiology and medicine. The LFP is the electric potential recorded in the extracellular space in brain tissue, typically using micro-electrodes implanted in specific cortical areas. LFP sample relatively localized populations of neurons, as these signals can be very different for electrodes separated by 1 mm or by a few hundred microns.

These neural data were collected while Non-Human Primates (NHP's) performed reward retrieval tasks involving reaching and grasping.

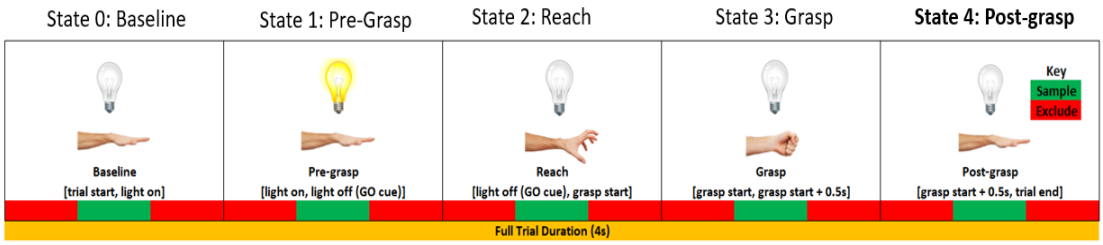



*Figure 4: Timeline of a single trial*

The data set is divided in trials. Figure 4 shows the structure of a trial, which starts in a baseline condition, once the visual cue was active, it would initiate the movement towards the handle, afterwards if would reach for it, to grasp it. Finally, the primate would release the handle expecting a reward.

The data we will be using has been separated by motor state (5 states in total, as seen in Figure 4) and it includes data in which the handle to reach for was positioned in different orientations (0, 45, 90 and 135 degrees). The dataset consists of the LFP collected from 128 electrodes, sometimes referred to as channels, and it is separated by trial. Each trial has a duration of roughly 0.25 seconds (508 samples). Moreover, there were 140 trials in total resulting in an input matrix of size [128,508,140,5].

The data used for the Left Vs Right problem has 10 states, 5 for the left arm and 5 for the right arm, so it will be a matrix of size [128,508,140,10].

## 3.2. Data Pre-processing

We will be using two different datasets, one for each test. As mentioned above, the dataset for the Five-State Classification will be of shape [128,508,140,5], whereas the dataset for the Left vs Right test will be of shape [128,508,140,10]. For the latter, the first 5 trials are for the left arm and the next 5 for the right arm. Basically, the first 5 states are the corresponding ones for the Five-State Classification test.

In order to normalize the data, we first compute the absolute value for all the trials to get rid of the negative values.

One extra step with the data will be performed after is has gone through the network, and that will be to use only the mean of all 128 channels for each trial and state. In other words, a full trial consists of 128 channels, 508 readings and 5 states, by taking the mean of the readings we are removing the temporal component of the data. This yielded a significant improvement in the performance of the model. The data that will use for the regressor will be of shape [128 x 5 x n_trials].

### 3.2.1. Frequency Bands

To analyze performance of different brain functions, we will be filtering the data in different frequency bands, known to encode different brain processes, so we will be filtering by the following frequency bands:

| Frequency bands | Hz |
|---|---|
| Baseline | 4-500 |
| Theta | 4-7 |
| Alpha | 7-15 |
| Beta | 15-30 |
| Low gamma | 30-70 |
| High gamma | 70-100 |
| Low ripple | 100-150 |
| High ripple | 150-200 |
| Low multi-unit | 200-500 |

*Table 1: List of Frequency Bands*

We will be using a battery of bandpass filters, to isolate each of these frequency bands.

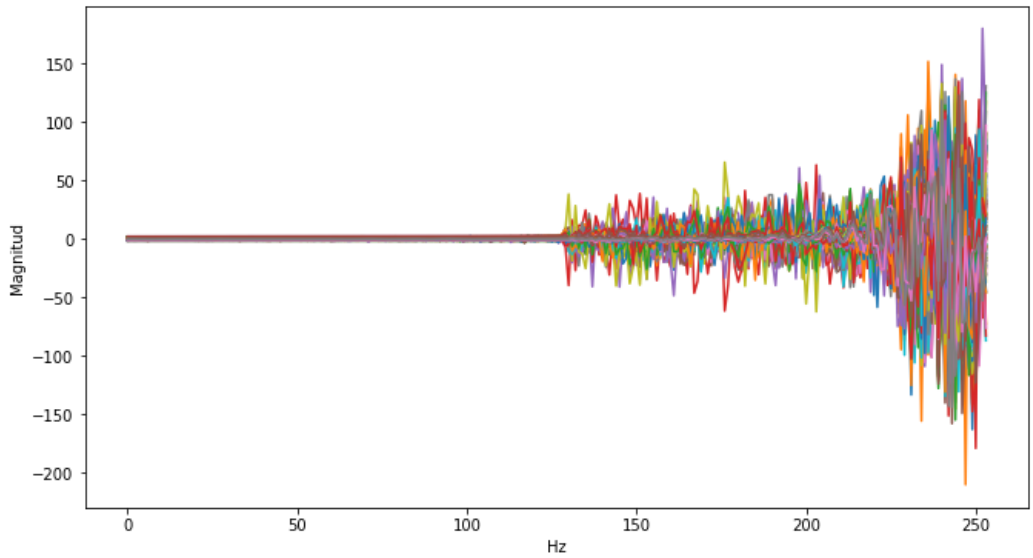



*Figure 5: Sample of unfiltered data*

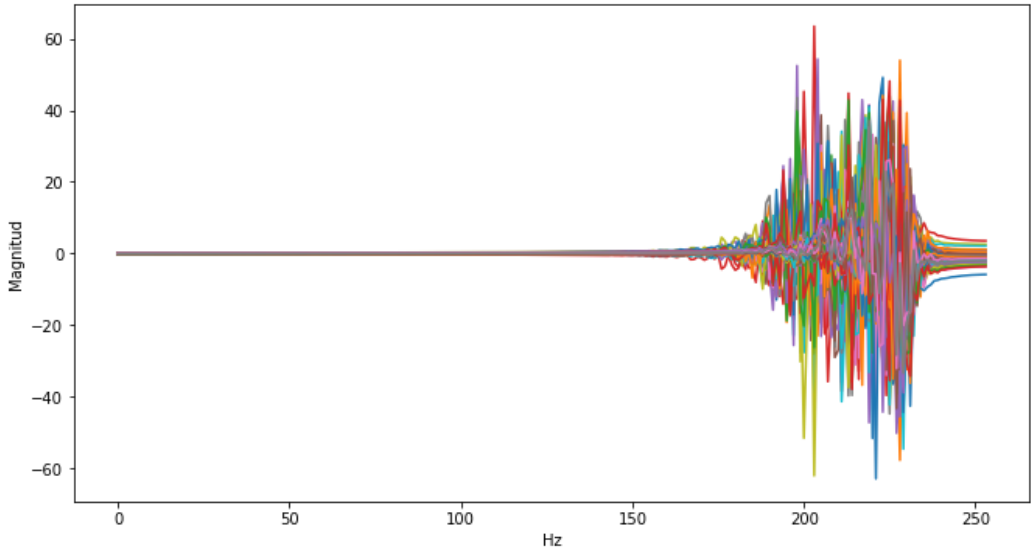



*Figure 6: Sample of data after low multi-unit filter*

On one hand, Figure 5 shows the Fast Fourier Transform representation of a non filtered recording, from a single state on a single trial. On the other hand, Figure 6 shows the same recording, after undergoing a low multi-unit bandpass filter. As we can see, we are only left with approximately the frequencies between 200 and 500 Hz.

## 3.3. Our reservoir computing network

Extending from the network presented in section 2.2.1, we will demonstrate how we build the different layers and weight matrices. This process is similar for both tests.

Firstly, the input matrix ($\mathbf{W}_{in}$) will be of size N×K, that is, the number of internal units and the number of input units. The data consists of 128 input channels. This means the value of K will be 128 as well. As a starting value, we will use 50 internal units. This is a hyperparameter that we will adjust later for better performance. So, to start, $\mathbf{W}_{in}$ will be a matrix of size 50x128.

The connections between the units will consist of a 10% probability of connection and their weights will be defined by a uniform distribution between [-1,1).

Secondly, the internal matrix ($\mathbf{W}$) will be of size 50x50 (NxN) as we have defined that we will be using 50 internal units.

The connections of the internal units will also have a 10% probability of connection, but it's weights will be defined using a Gaussian distribution with a mean of 0 and standard deviation of 1.

Thirdly, in our problems we won't be using feedback connections as they did not improve the performance, so $\mathbf{W}_{back}$ doesn't need to be defined.

Finally, the output matrix ($\mathbf{W}_{out}$) will be fully connected to the inner matrix, and its weights will be defined through the regression method of our choice. The size of $\mathbf{W}_{out}$ is of L×N, where L is the number of output units. If we have 5 possible states, we need 5 output units, so the final size of $\mathbf{W}_{out}$ will be 5x50.

As we won't be using the $\mathbf{W}_{back}$ matrix, the state update equation will be defined as:

$$\mathbf{x}(n+1)=f(\mathbf{W}\mathbf{x}(n)+\mathbf{W}_{in}\mathbf{u}(n+1))$$

We will also check that the largest eigenvalue ($|\lambda_{max}|$) is $|\lambda_{max}| < 1$, as mentioned in section 2.2.2, that is a key concept in order to ensure we have Echo States.

### 3.3.1. Hyperparameters

In Machine Learning and Deep Learning, the hyperparameters are those parameters whose values are set prior to the training process, for example, the number of nodes in the reservoir layer, or the probability of connection between layers. These are the ones which we will try to optimize by performing a grid search.

Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model that is being trained.

A grid search is an exhaustive process in which we search for the optimal hyperparameters of the targeted algorithm. More specifically, we will go through a previously chosen subset of hyperparameters, looking for the ones that yield the best performance.

### 3.3.2. ESN Exploitation

For all the tests that we will perform, 80% of the data will be used for training, leaving the remaining 20% for testing.

After having everything set up, we will be able to train the network and start doing predictions. As mentioned above, we use 80% of the data for training, that is, 112 trials, leaving 28 for testing. All the accuracies given will be from predictions made from tests with the testing data, so trials that the network has not seen before.

We will try to improve the accuracies by means of optimizing hyper-parameters and applying different band filters to the data.

# 4. Results

In this section we show the results of the tests we performed.

## 4.1. Classifying Five Consecutive Movement Related states

We first classified five consecutive movement related states (see section 3.1). Since the states are consecutive, it will be important to preserve this order when training and testing.

We will be using the network proposed at section 3.3. although there may be modifications in order to improve performance.

Let's start by doing a test without the RC model and no filters, to see how the classifiers perform, we will use those results as a baseline to improve from.

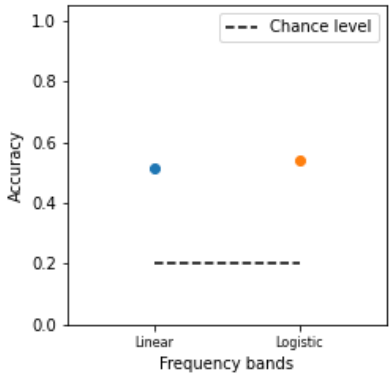



*Figure 7: Results without Reservoir Computing*

We can see that both Linear and Logistic Regression perform well above chance level, with their mean between 50 and 55% respectively. Although these results are not bad, they can be massively improved with the help of Reservoir Computing.

To demonstrate this fact, we will perform a test using Reservoir Computing with raw hyperparameter values:

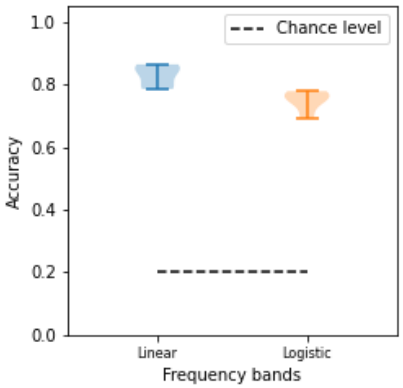



*Figure 8: Results with Reservoir Computing*

It is clear that both results have been massively improved, reaching a mean of approximately 80% with Linear Regression and 77% with Logistic Regression.

Now that we have a clear example of the potential of RC, we will try to further improve the performance of the model by optimizing the hyperparameters.

### 4.1.1. Hyperparameter optimization

As mentioned in section 3.1.1, we optimized the number of nodes inside the reservoir, the probability of connection of the input layer and finally the probability of connection of the reservoir layer.

We will start with the number of nodes, as it is the simplest hyper-parameter to test, we will be trying 1, 5, 10, 20, 50, 100 and 200 nodes.

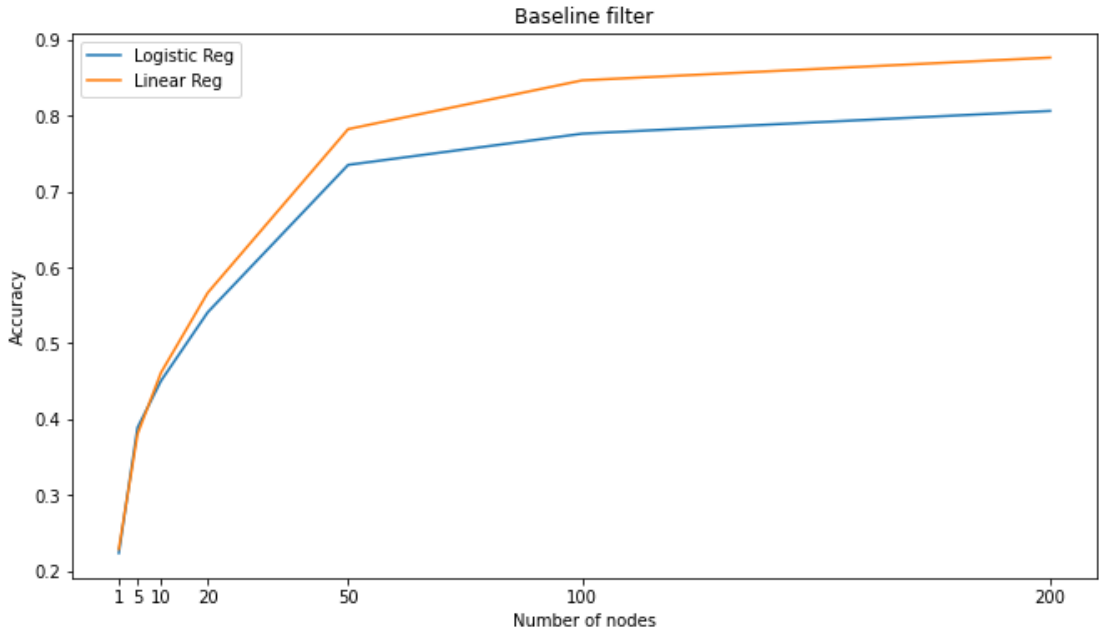


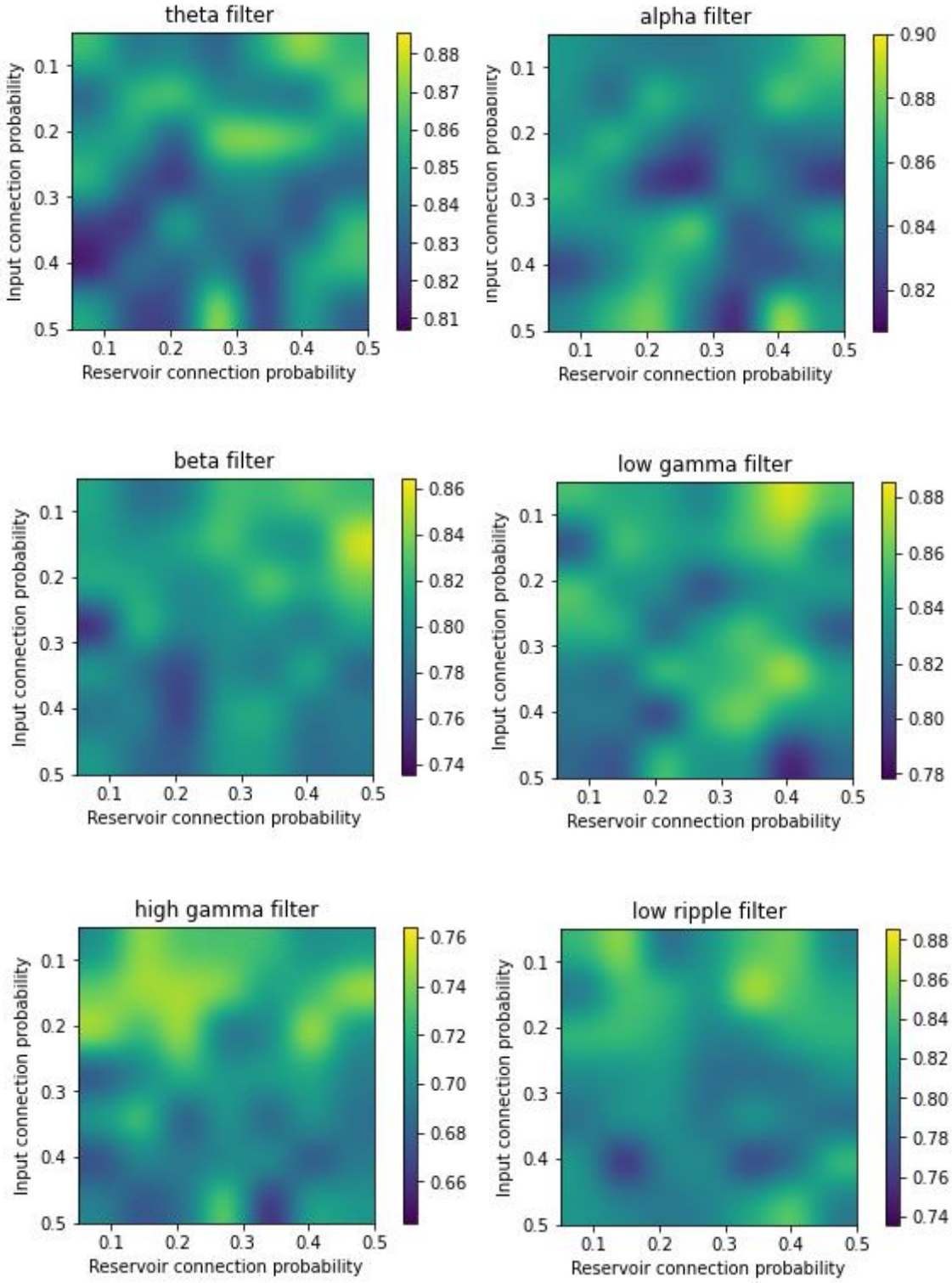

*Figure 9: Number of nodes optimization*

Figure 9 shows that the top performance is asymptotically reached at 200 nodes. However, we used a 100 node reservoir, as it retains a close to optimal performance while speeding up the training process.

We also optimized both the input and reservoir connection probability by doing a grid search with the following values: [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]. We will not be going higher as we want to keep the reservoir matrix as sparse as possible.

These probabilities define the likelihood of each node to be connected with one in the next layer, in case of the input layer, or with nodes within the same layer, in case of the reservoir layer.

This grid search will be performed for each specific frequency band.


theta filter
alpha filter
beta filter
low gamma filter
high gamma filter
low ripple filter
Input connection probability
Reservoir connection probability

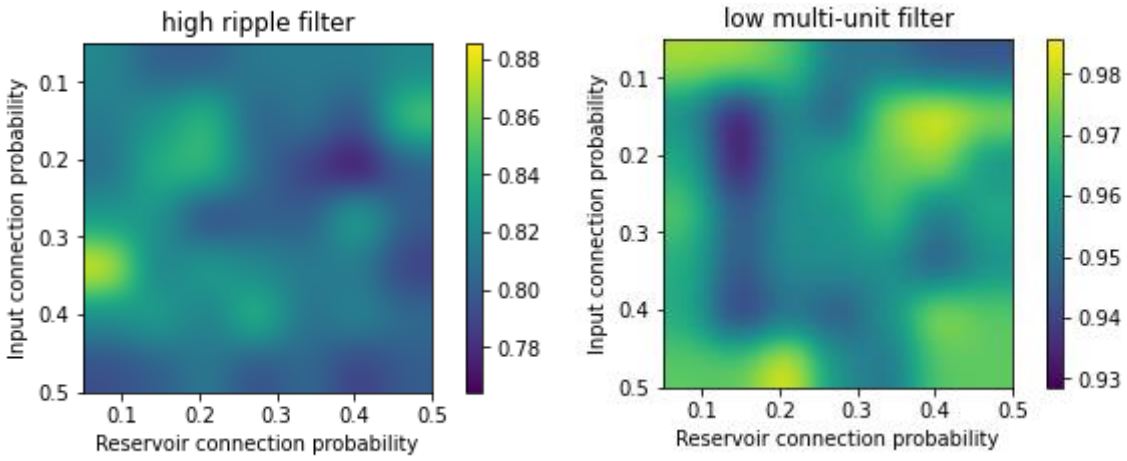



*Figure 10: Grid search heatmaps yielding accuracy*

Figure 10 shows impressive accuracy values for some bands, for example, low multi-unit reaches approximately 98.5% accuracy. Although these results are excellent, the heatmaps show different optimal hyperparameter configurations for each frequency band filter. Therefore, we are most interested in the optimal configuration across all frequency bands.

In order to determine this configuration, we will take the mean of all the heatmaps at the same time:

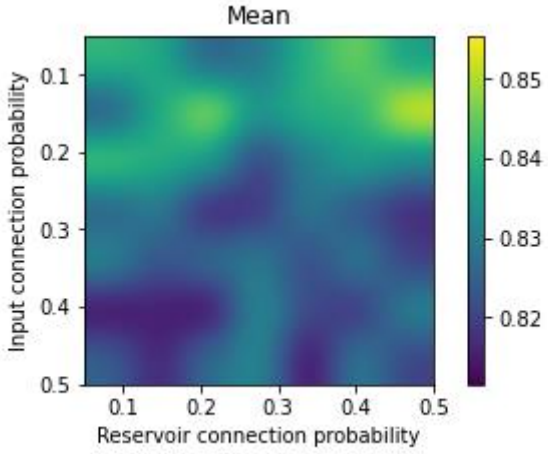



*Figure 11: Mean of heatmaps*

We now see a much clearer heatmap, although the mean values are not as high, we will still get good results when executing the model for each band. The optimum values seem to be 0.15 input probability connection and 0.5 reservoir probability connection.

Finally, we will have a look at the performance of the model using the optimal hyperparameters, for all the frequency bands and for both classifiers:

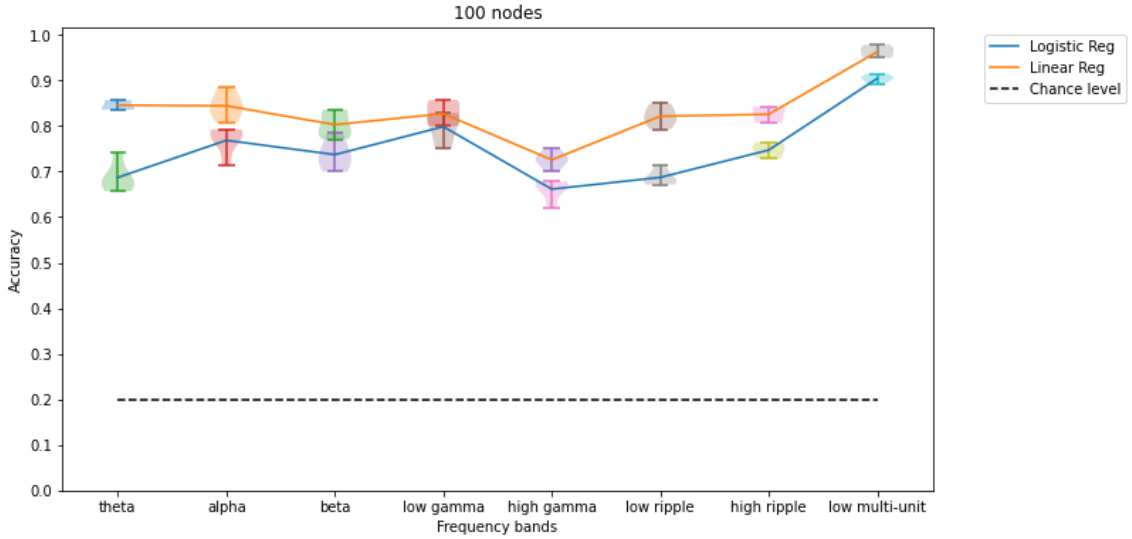



*Figure 12: Results after hyperparameter optimization*

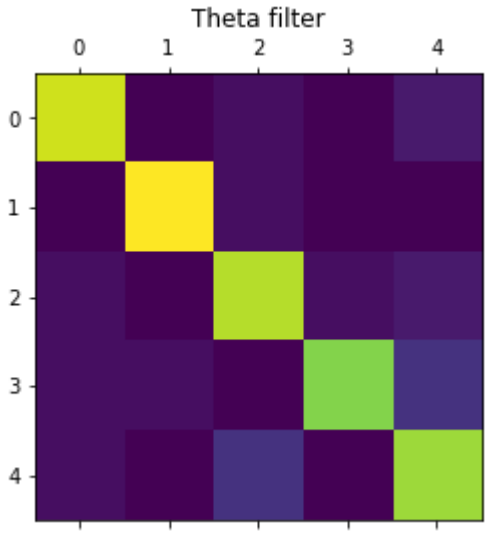
We can now see we are still getting good results even with the same hyperparameters for all the frequency bands. Performance peaks using the low multi-unit data reaching approximately 95% accuracy for the linear regressor and 90% for the logistic regressor.

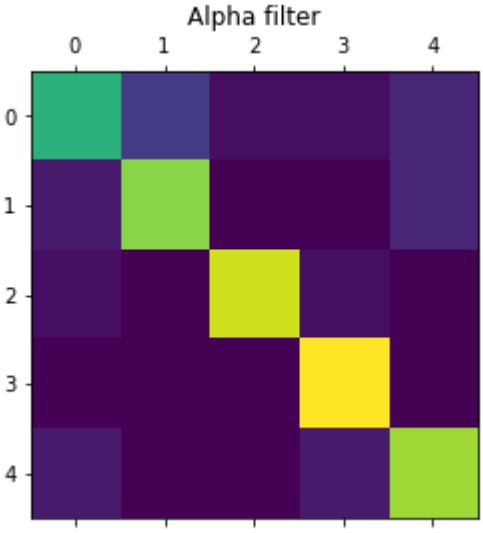
### 4.1.2 Tests

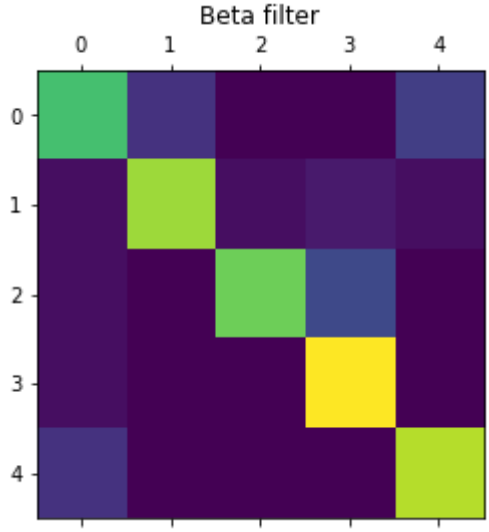
Now that we have a fully working model with good results, it is of interest to perform some tests to try to prove that we have achieved what we wanted.

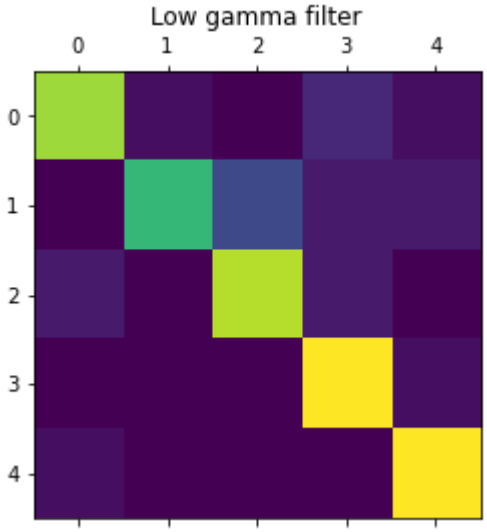
Recall from section 1.1, that we intended to classify all five states from a movement related sequence of actions. Although we could argue that we already prove that with reaching such high accuracies, we do not really see how well each state is identified.

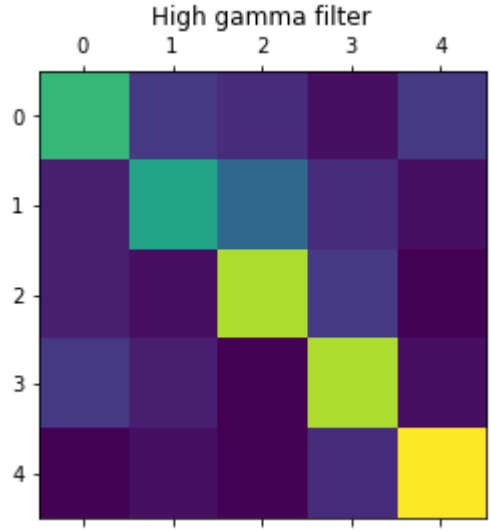
To that end, we will compute confusion matrices, those will show us how accurately each state is classified for each frequency band. We will be performing the tests with the linear regressor as it yielded better results than the logistic regressor.

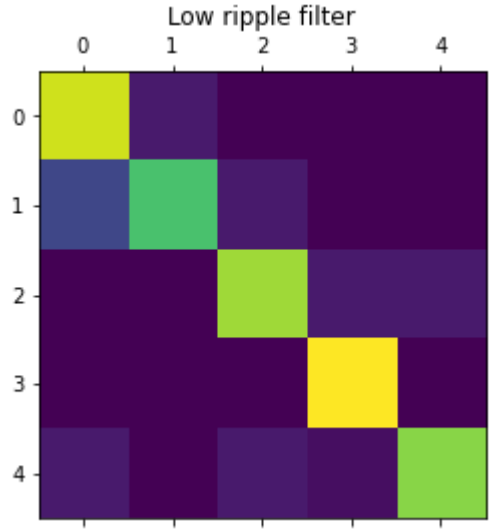

Theta filter
0 1 2 3 4
0 1 2 3 4


Alpha filter
0 1 2 3 4
0 1 2 3 4


Beta filter
0 1 2 3 4
0 1 2 3 4


Low gamma filter
0 1 2 3 4
0 1 2 3 4


High gamma filter
0 1 2 3 4
0 1 2 3 4


Low ripple filter
0 1 2 3 4
0 1 2 3 4

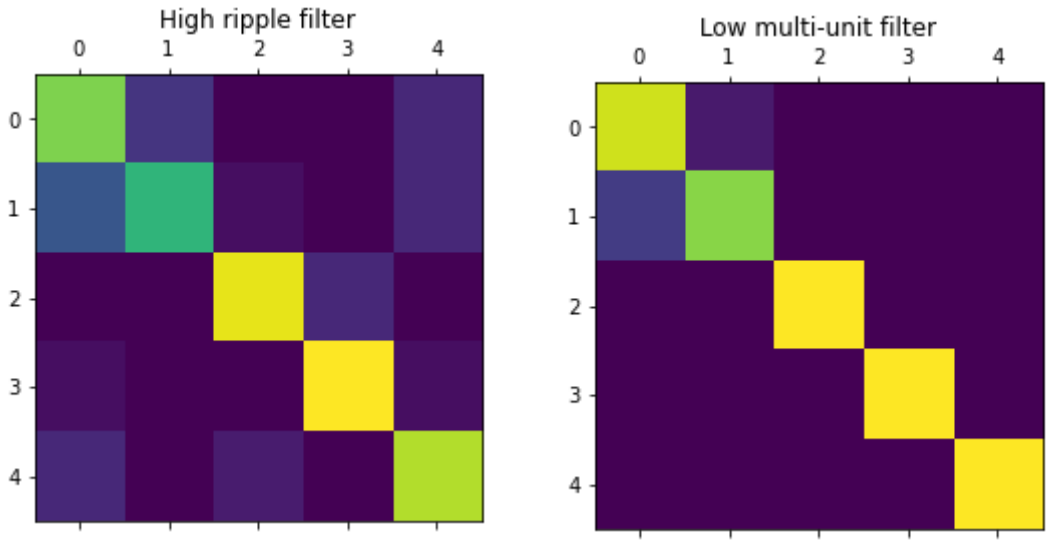



*Figure 13: Confusion matrices*

As we can see, we are classifying all the states very well for all the frequency bands, especially with the low multi-unit band. With these matrices we can see that the most distinguishable state is the 3rd, meanwhile the ones which appear to be confused more often are the 0 and 1 states. If we recall figure 12, we can see that higher accuracies show less confusion in their respective matrix.
We can also see that the states 0 and 1 are better classified when they have a lower frequency band filter applied, that could mean that both those actions have a richer encoding in lower frequencies.

We can safely say that we have achieved to classify the states, especially in higher frequencies.

We also hypothesized that running the data through the RC network would provide a richer encoding of the data and would achieve better classification.

In order to test this hypothesis, we compare the previously obtained results with the results of the data directly classified with the regressors. Note that the data has had the same processing in all cases, the only difference is whether they have either gone through the reservoir.

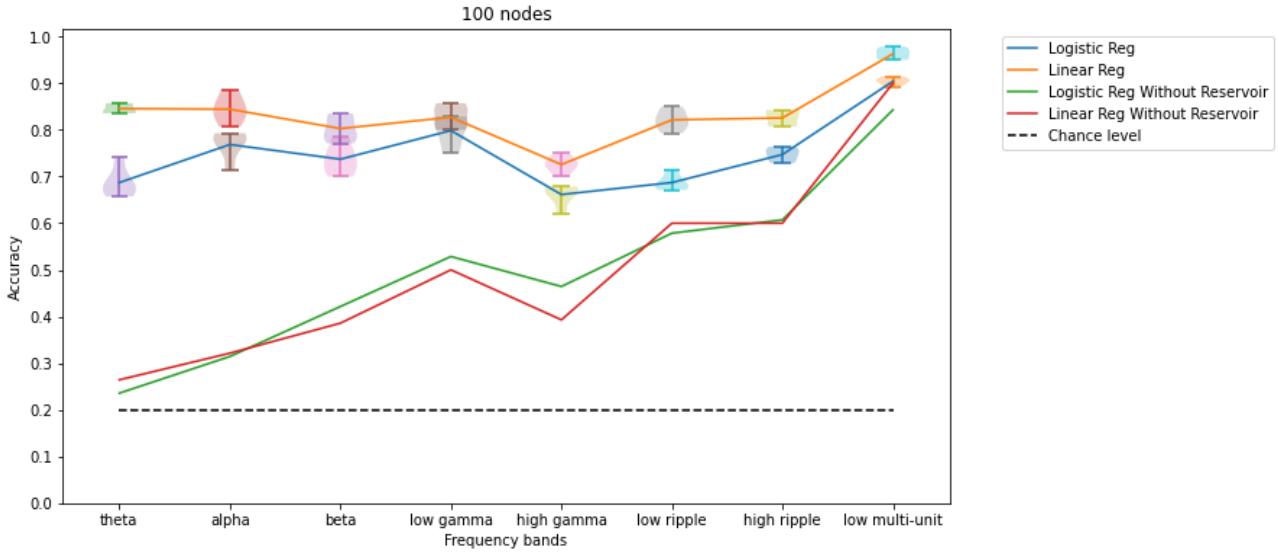



*Figure 14: Comparing results with and without RC*

We can see that classifications with Reservoir Computing are always better than classifications without it. Especially when the data is not so rich, like in lower frequencies, the RC model seems to really help by providing a richer encoding of the data, whereas when the data is already rich, the improvement is not so notable.

If we recall from section 1.2., we saw that using the MLR classifier with high frequency data yielded a score of 0.8, which is the same accuracy we are achieving with our test. It was also mentioned that lower frequencies resulted in worse scores, this behavior is also reflected in our test.

It does, in fact, appear that the RC model helps enrich the data encoding, although this may not be the case for all types of data.

## 4.2. Laterality states: Left vs Right arm

In this test we will be classifying when the NHP is using either the left or right arm. State is not taken into consideration.

We will be using the same workflow seen in the previous test, so we will start by using the raw network seen in section 3.3. but with a small modification. As this test is a binary classification, we only need 1 output node, so the $\mathbf{W}_{out}$ matrix will be of shape 1x50.
We will also see some modifications in the number of nodes and connection probabilities in order to improve performance.

Let's start as well by doing a test without the RC model, to see how the classifiers perform, we will use those results as a baseline to improve from. In this case, instead of using Linear regression, we will replace it with a 1NN classifier, so we will be using both the 1NN and Logistic classifiers.

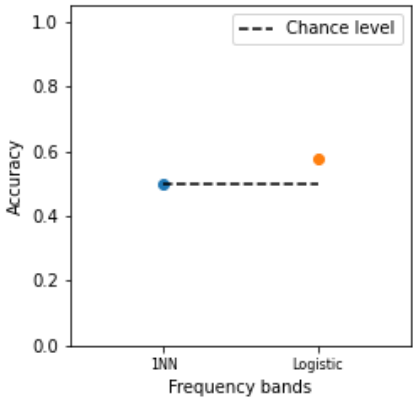



*Figure 15: Base test without RC*

We can see that now as we are doing a binary classification, the chance level is now set at 50%, we can also see how the 1NN is performing at that level, and that the Logistic regressor is performing at approximately 57% accuracy. Basically, the classifiers are not managing to extract relevant features from the data, and that is why the performance is so poor.

Let's check now how our network helps to classify the same data. We will use raw hyperparameter values again, as the ones we optimized for the past test may not be optimum now.

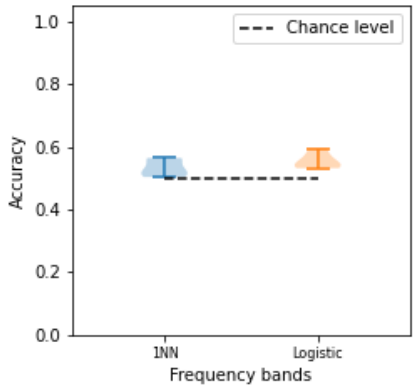



*Figure 16: Base test with Reservoir Computing*

We can see that we have slightly managed to escape chance level, but still, these results are not really what we would like them to be. As before, we will try to improve these results by optimizing our hyperparameters.

### 4.2.1. Hyperparameter optimization

As we have seen in section 4.1.1. and mentioned in section 3.1.1, we will be optimizing the number of nodes inside the reservoir, the probability of connection of the input layer and finally the probability of connection within the reservoir layer.

In the first grid search, we will try to look for the optimal number of nodes in the reservoir layer. We will perform the search with: 1, 5, 10, 20, 50, 100 and 200 nodes.

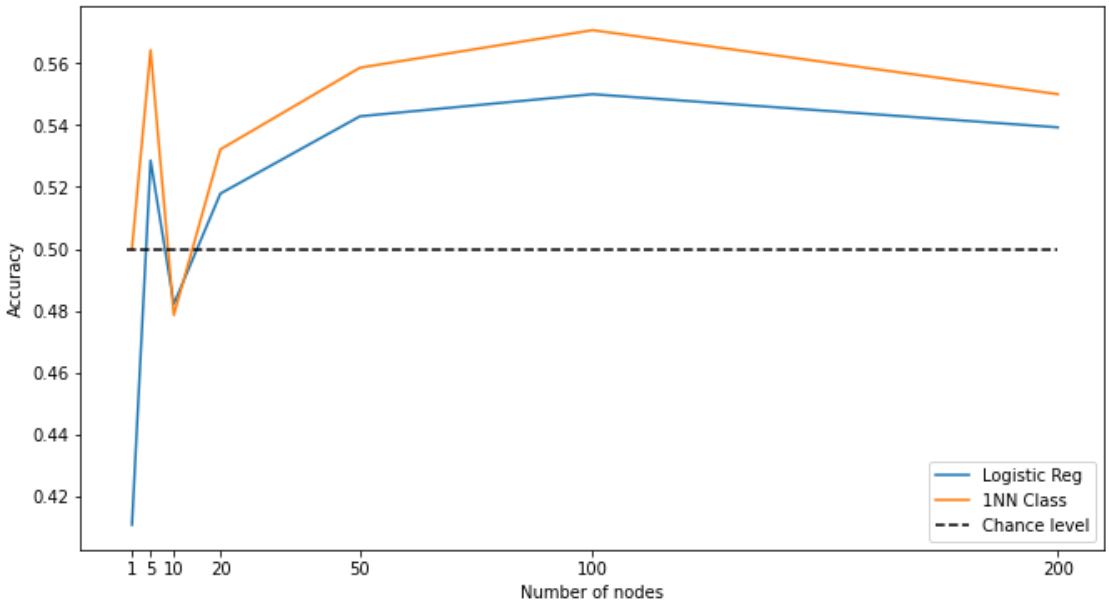
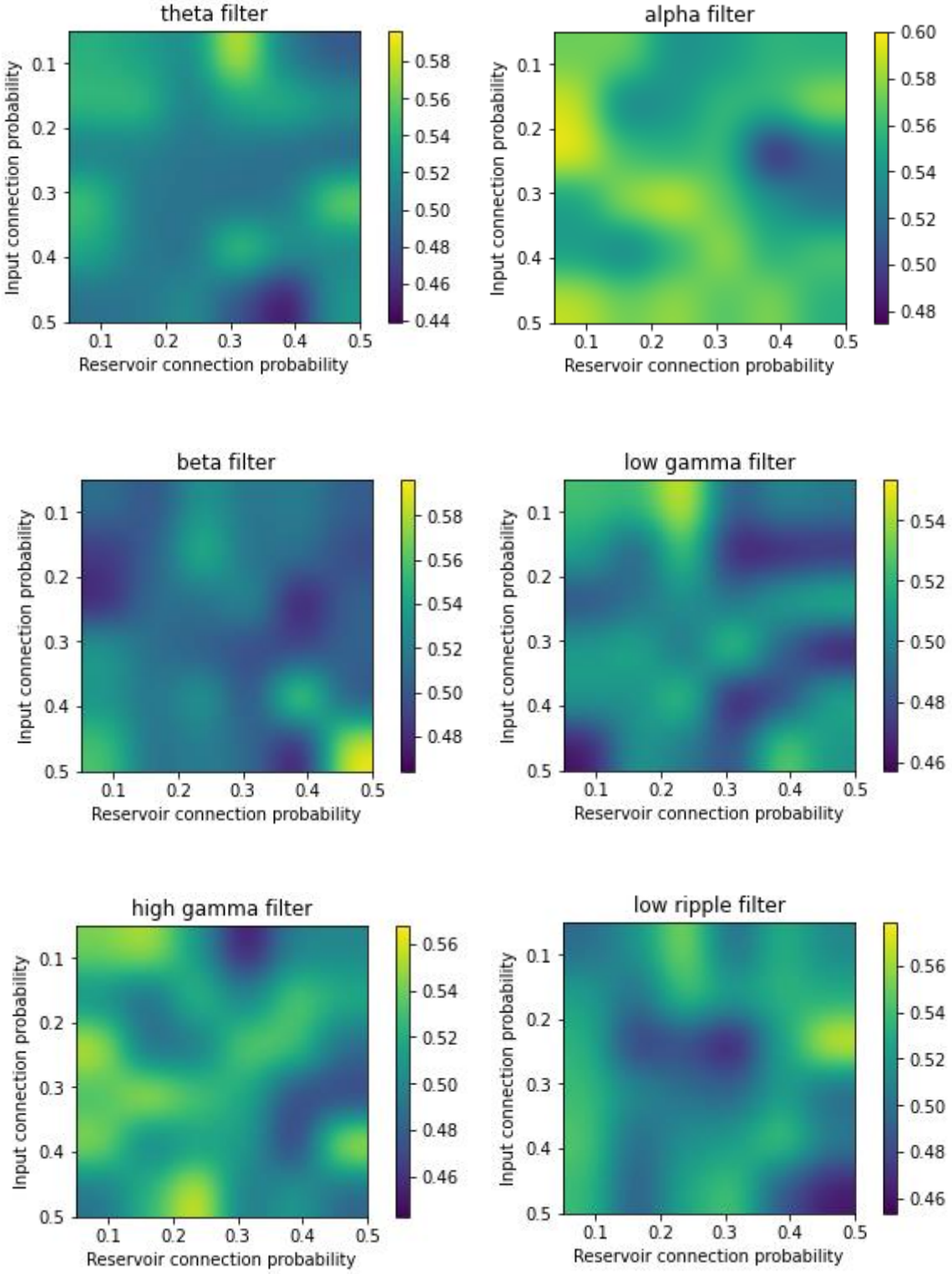



*Figure 17: Grid search for number of nodes*

As we can see in the plot, performance peaks at 100 nodes, although we do not really see a big improvement as we did before. We now know that we will probably achieve better results if we use this number of nodes.

In the next grid search, we will try to optimize both the input connection probability and the reservoir connection probability. Both probabilities will be tested with the following values: [0.05, 0.1, 0.2, 0.3, 0.4, 0.5].



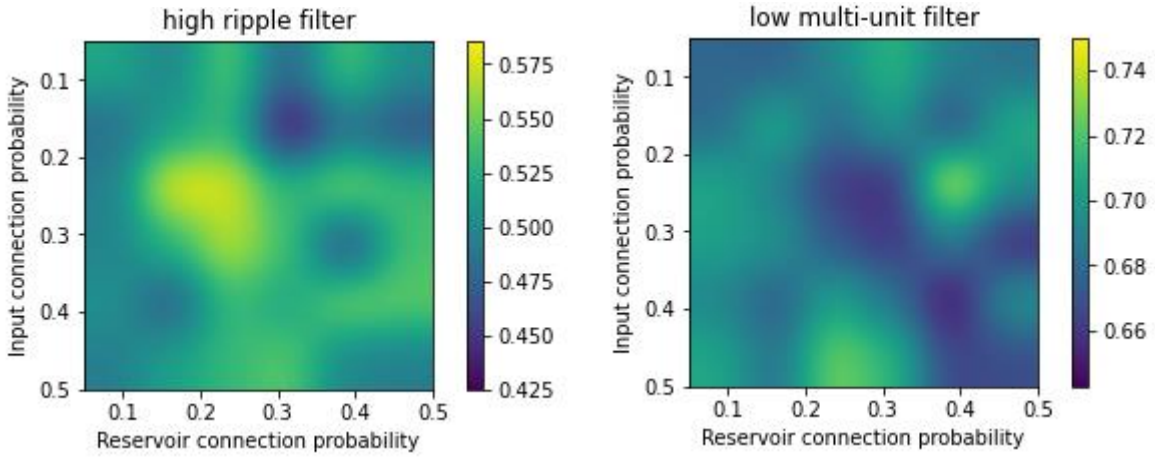



*Figure 18: Accuracy heatmaps LeftVsRight*

As we can see with the heatmaps, they all show different optimal setups depending on the frequency band they have been computed with. As before, we only want one global pair of parameters, so we will compute the mean of all the heatmaps.

Even though we see some good results, especifically with the low multi-unit filter, reaching close to 75%, the mean of all the heatmaps will be a good compromise so all the frequency bands can perform decently.

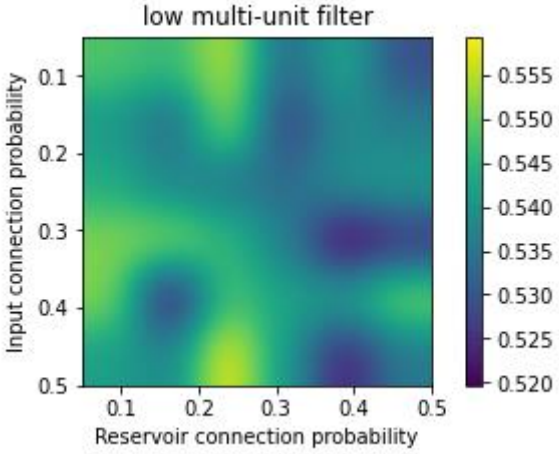



*Figure 19: Mean heatmap LeftVsRight*

We can see that we have more than one “higher accuracy” spots, but the one at the bottom in the middle appears to have the highest value of them all. That is, 0.5 input probability and 0.24 reservoir connection probability.

Now we will use the optimized hyperparameters to see how well the model performs.

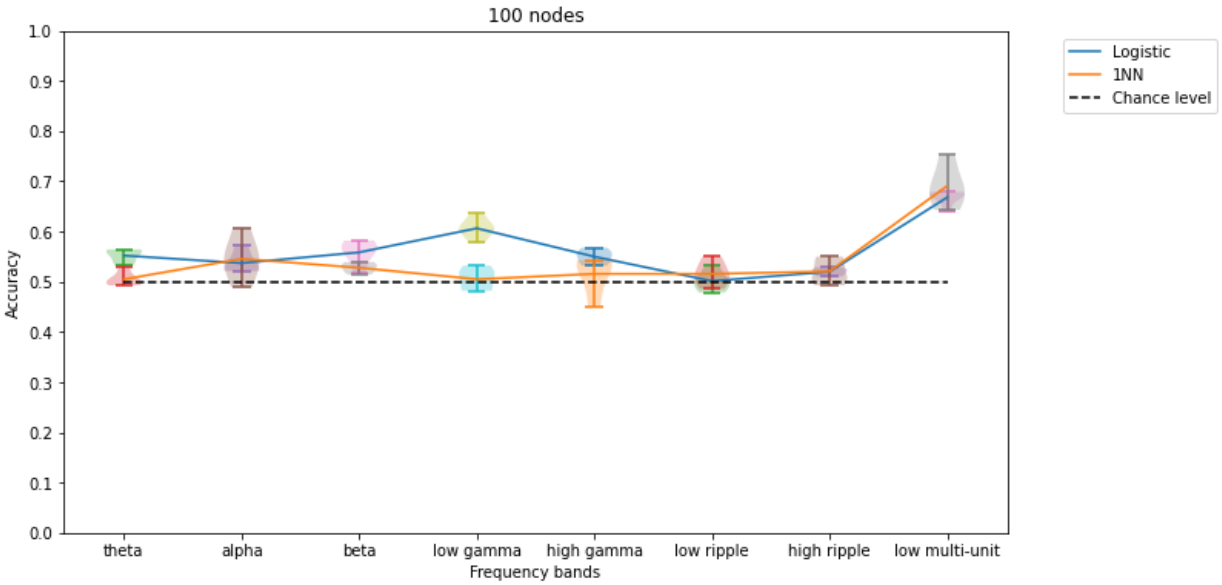



*Figure 20: Results after hyperparameter optimization LeftVsRight*

We can see how we are only getting decent results with the highest frequency band filter, with approximately 70% accuracy for both the Logistic regression and 1NN classifiers, with the latter having its highest value at 75%.

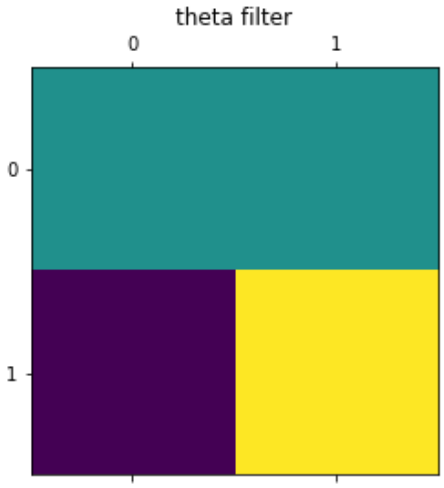
Seeing these unimpressive results, we can assume that the data does not encode much information of which arm is performing the action. Although we are getting some fairly good classifications at high frequencies, if there was more information of the arm performing the action in the data, we would probably see performance resembling that of the Five State Classifier.

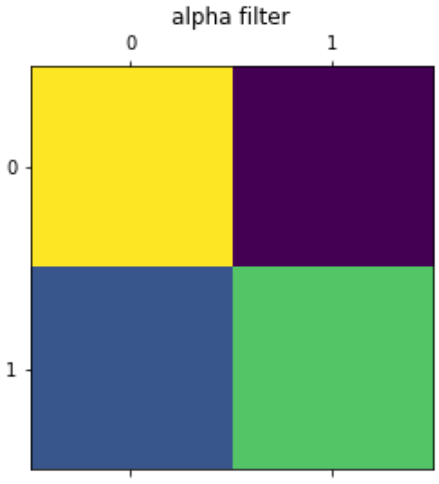
Despite that, we are still going to perform the tests, to see if we can extract some ideas on how to improve the results.

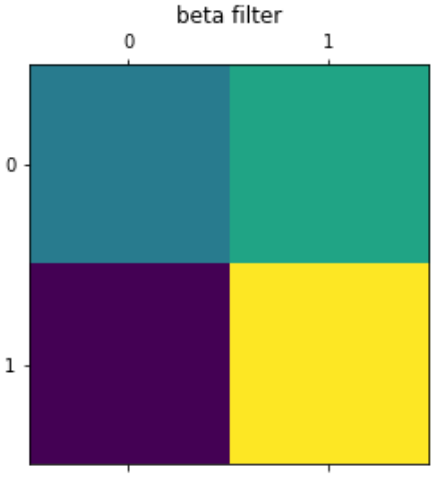
### 4.2.2. Tests

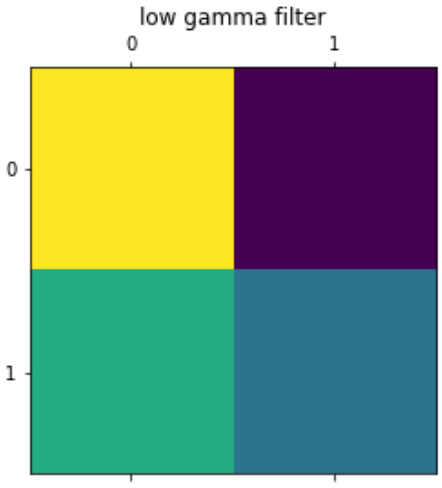
Following the workflow from section 4.1, we are firstly going to see how well the states are differentiated, which frequencies produce the best results, and which states are misclassified the most.

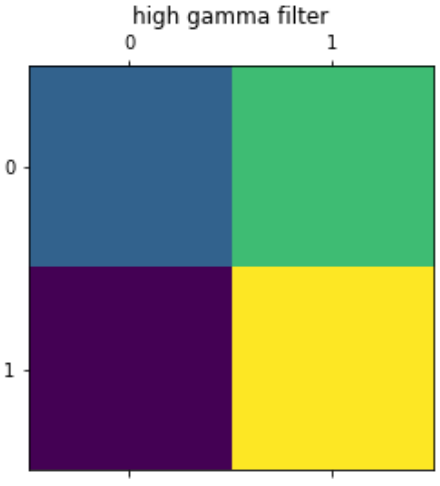
In order to do so, we are going to build confusion matrices for each frequency band.
In this case, we will only use the 1NN classifier as it has the highest performance value in the low multi-unit filter.

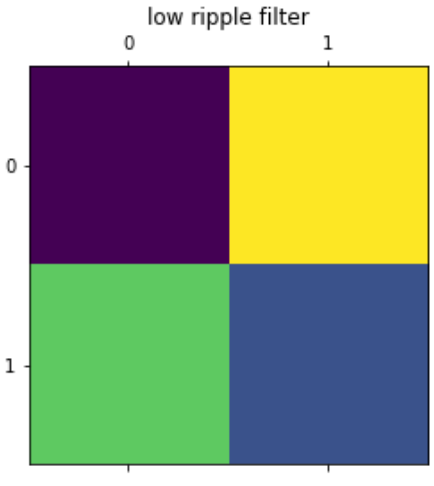

theta filter
0
1
0
1


alpha filter
0
1
0
1


beta filter
0
1
0
1


low gamma filter
0
1
0
1


high gamma filter
0
1
0
1


low ripple filter
0
1
0
1

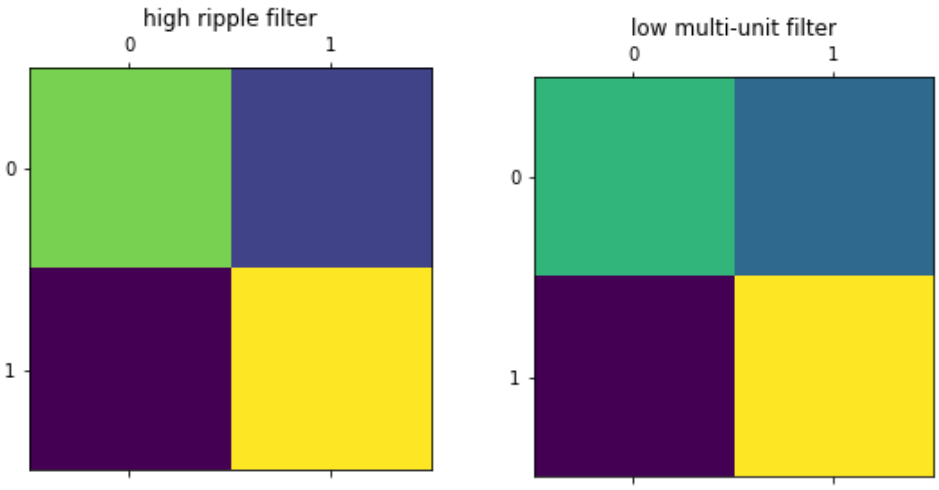



*Figure 21: Confusion matrices LeftVsRight*

For clarification, the color yellow represents a higher value and the darker it gets, the smaller value it has.

Although it seems like these are just random values, and that we are not really classifying much, we can tell from looking at this that state 1 (Right arm) shows better classifications than the state 0 (Left arm). We can also see that state 0 is more often classified as state 1. The opposite case doesn't happen nearly as frequently. This could mean that the data has a stronger encoding for the right arm, thus making it harder to classify the left arm. This actually makes sense since when the test with the NHP was performed, 96/128 electrodes were in the left hemisphere of the brain. Also, it is known that the left hemisphere controls motor actions of the right side of the body.

Though we are only getting decent results at high filtering frequencies, we are still managing to classify the data at above chance level.

Finally, we will look if we are managing to improve the classification task by using reservoir computing for all frequency bands.

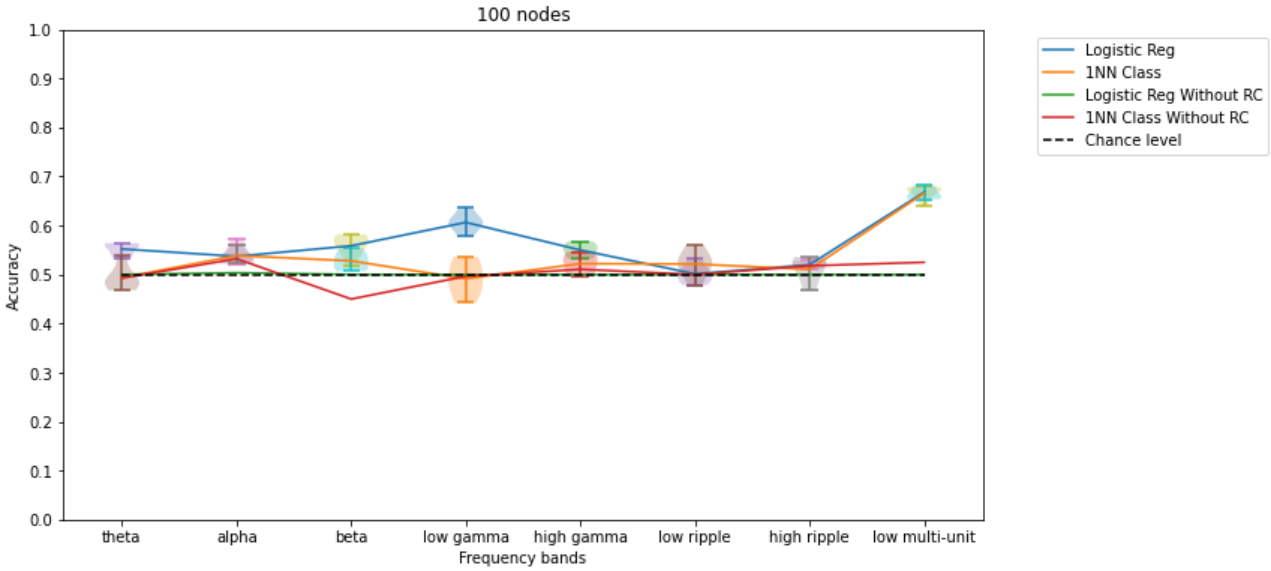



*Figure 22: All results LeftVsRight*

As we can see in the plot, the only results that manage to get well over chance level are the ones which have gone through the RC network. This further proves that our model is helping to enrich the data in order for it to be better classified.

Although the results from this test do not seem good, we have to keep in mind that the data used was gathered in order to visualize the states of the NHP through a simple decision making task. So the goal of data task was not to differentiate between left and right arm, so it does make sense that this classification task has poor results when compared to the Five State Classification.

Despite that, we were still able to classify both left and right arms decently, which goes to show the potential of Reservoir Computing in the Computational Neuroscience field.

# 5. Discussion

The goal of this study was to identify different movement related states from a neural dataset recorded from a non-human primate, and to see if the representation of the data after going through the model would help classify the five movement-related neural states.

We started by introducing the current state of Recurrent Neural Networks, how they work and what are they used for. Afterwards, we saw the differences between standard RNNs and Reservoir Computing. We dug deeper into RCs, showing how they are built and some of the mathematics behind them. We also talked about Echo State Networks, a key concept behind RC, and introduced its advantages compared to RNNs.

Secondly, we presented the experiment where the dataset came from, we saw in detail the simple task the NHPs performed and how this data was presented to us. We also showed the different processes the data would undergo in order to make better classifications.

Thirdly, we talked about how our neural network would be built and all the modifications that later would be applied to it in order to improve our results.

Finally, we performed two different tests in order to prove our initial objectives. The first tests consisted of training the model so it could separately identify the five movement related states. Not only did we manage to do so with impressive accuracies, we also proved that the classifiers yielded higher classification accuracy after using our RC model.

The second tests consisted of distinguishing left and right movements of the arm. Although we found considerably worse results compared to the first test, we still managed to classify the data at above chance level for high frequencies, and also saw that the classifiers were working better when using our RC model.

## 5.1 Limitations

Since we found good results in our tests, it would have been interesting to further test the model in order to see how useful RC can actually be, not only in the Computational Neuroscience field but in others as well.

Due to the short deadline we had, it was impossible to develop and perform more complex tests to see how the model would behave. This would have been interesting as we would have managed to dig further down into the Computational Neuroscience field and hopefully achieve results that would help future investigations.

As previously reported by all Artificial Intelligence projects, large computational power and long hours of processing are required to train the models. Although RC is more efficient, we still had some limitations as we could not really increase the number of hidden nodes or perform data augmentation to further train the model. Despite that, we still had really good results, but it would have been interesting to see the behavior of the model in more extreme cases.

## 5.2 Future work

Some ideas of what could be done with what we currently have are:

- Perform more and more complex tests:

    1. Once we have a working structure, remove different nodes from the network to try to find the more significant ones. Once found, see how they behave and study why are more significant.

    2. Instead of using the mean of all the trial for each state, compute the mean every 50 or 100ms, and check how the accuracies are affected.

    3. Use different classifiers.

- Use different datasets, RC is very good at predictions, it would probably perform well in predicting stock market prices or for tasks in other fields as well.

## 5.3 Conclusion

In the end, the results showed that we achieved objectives we were aiming for. We proved that Reservoir Computing is able to accurately classify movement related states from a Local Field Potential dataset, and also to decently distinguish between the left and right arm with the same dataset in higher frequencies. Moreover, we proved that RC is able to provide a better classification for the mentioned tests, compared to just using classifiers by themselves. All of the above was done while exploring a quite new and interesting technique, and showing the big potential it has in the Computational Neuroscience world.

# Bibliography

Maass, W., Natschläger, T., Markram, H.: Real-time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. Neural Computation 14(11), 2531–2560 (2002)
Google scholar

Jaeger, H.: The "Echo State" Approach To Analysing And Training Recurrent Neural Networks. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)
Google scholar

Caton R.: The electric currents of the brain. BMJ 2:278 (1875)
Google scholar

Mattia Rigotti, Daniel Ben Dayan Rubin, Xiao-Jing Wang and Stefano Fusi.: Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. Frontiers in computational neuroscience. (2010)
Google scholar

Enel P, Procyk E, Quilodran R, Dominey PF.: Reservoir Computing Properties of Neural Dynamics in Prefrontal Cortex. PLOS Computational Biology 12(6): e1004967. (2016)
Article

Michael DePass, Ignasi Cos: Characterization of Movement-related Neural States in NHP's. Master Thesis at University of Barcelona, Facultat de Matemàtiques I Informàtica. (2019)