# Comparison of cognitive strategies used by human subjects and recurrent neural networks to categorize visual sequences

Nicolas YAX

June 16, 2021

## 1 Introduction

Classifying data from imperfect information is probably the most classical problem in Artificial Intelligence (AI) but also in neurosciences (for a review on the issue, see., e.g., [DJ12]). It is one of the main questions from the last decade in cognitive sciences resulting in the creation of sequential integration models (for a review in cognitive psychology, see, e.g., [Rat04]; for a review in cognitive neuroscience, see, e.g., [GS07]).

From a cognitive science point of view what is really interesting in such tasks is how people or animals classify data. Therefore researchers investigated representations developed through training and testing in order to have a better understanding of structures involved. When comparing human performance to the performance of AI systems, researchers have focused in particular on the visual classification of images ([DJ12]). This research has shown that deep convolutional neural networks designed to resemble the architecture of the visual cortex are capable of achieving high performance in difficult image classification tasks - close to human performance in some situations. However, these networks are typically only capable of solving only one image classification task at a time (e.g., classifying images with respect to whether they contain a dog or not), whereas the human brain is capable of flexibly solving many tasks based on different rules and strategies - using contextual 'cues' to select one rule over the others depending on one's current goal (for a recent example using a standard task from cognitive psychology, see, e.g., [MN14])

## 2 Context

This project is focused on the format of representations used by human subjects when performing a controlled visual classification task based on recent work in the group of my supervisor (see, e.g., [DK16]; [WW21]). The key question addressed by this project concerns whether human subjects integrate visual information for categorization in a stimulus-dependent format (in a format determined by the type of stimuli being processed, that is, their dimensions such as orientation and color) or in a category-dependent format (in a format determined by the different categories between which one has to choose). Recent work in the group of Valentin Wyart (my supervisor) has shown that it is the integration of visual information that is the limiting factor for performance - i.e., the main source of suboptimality - during image classification ([DK16]). It is thus important to know the format in which this integration process is being performed in such tasks. The task used to study this process is typically of psychology, laboratory-based experiments: it uses controlled stimuli and conditions that are not typical of everyday tasks outside the laboratory, but useful because we can control all aspects of the task and thus understand more precisely the different cognitive processes used by tested subjects to solve the task.

We tried to answer this question first by an experimental cognitive approach which has been tested on humans to determine if people use the stimulus-based or the category-based format of representation. Secondly, we used a simulated process involving artificial neural networks aimed at studying architectures able to reproduce human data and to study representations developed by those artificial agents. Julie Drevet (PhD student in the group) worked on the cognitive aspect of the project in human subjects. My contribution was on the simulation part with artificial neural networks. The primary goal of the project was to understand whether and how well recurrent neural networks can

solve the task performed by the human subjects, and to understand the representations used by the neural networks to solve the task.

In practice we haven't been able to get all the human data before the end of the internship. Therefore, my main job was to train neural networks on the task and to try to get results similar to human classification accuracy by exploring as many neural architectures as possible as well as understanding how these structures learn and work. This work involves appreciating how data are processed by the network in addition to classification errors.

# 3 Background

## 3.1 Neural Network

Biological neural networks inspired artificial neural networks. These structures can efficiently process data given from input neurons to output neurons. For example, walking requires visual stimuli as well as equilibrium which come from specific sensory neurons. A multitude of neurons then processes this information up to motor neurons. Those neurons can produce a muscular answer making people able to move. However, biological and artificial neurons are still quite different, starting from how people name their components.

Biologically speaking people call "layer" a group of neurons that have similarities. They are usually involved at the same stage of a process (color processing in visual areas for example). Those biological neurons can be connected between layers as well as inside the same layer.

In artificial neural networks, vectors of values represent layers of neurons. The n-th value is the activation of the n-th neuron. Connections between neurons are very often matrix multiplications which makes it possible from the input vector (activation of input neurons) to compute the output vector (activation of output neurons). Staking many layers enables neural networks to compute more and more complex functions.

However in artificial neural networks it is the connections which are called *layers* instead of vectors. Thus it is important when working between artificial and biological neural networks to understand the difference between biological layers and artificial layers. As a computer scientist I will use the term *layers* to mean artificial layers but because I worked between both fields I will regularly use layer when speaking about those activation vectors as it is more visual.

### 3.1.1 Layers

In artificial neural networks, layers can be seen as operators which transform an input vector $v_i$ into an output vector $v_o$ using non-linear mathematical operations. Here are few examples of layers we will use later:

### 3.1.2 Fully Connected Layer (FC)

This is the most common layer. FC stands for *fully connected* which means it represents connections between 2 layers of neurons where each neuron from the previous layer is connected to all neurons of the next layer. Mathematically here is how it is computed :

$$v_o = f_a(W.v_i + B) \tag{1}$$

with $v_o$ the output vector, $f_a$ the activation function, $W$ a weight matrix, $B$ a bias vector and $v_i$ the input vector.

The weight matrix represents weights of connections between neurons and the bias vector makes it possible to have various activation sensitive neurons. This means some neurons will easily be activated while others will need to be more stimulated to activate. It is the activation function that will express what activation means in the network. The most common activation functions are :

**tanh**

$$f_a(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$

Tanh (Fig 1) is the closest activation function to biological neurons as once a biological neuron is excited enough (enough input stimulus compared to its threshold) it will send a spike. Spikes have

constant amplitude and duration which means it encodes a 1 where no spike encodes a 0. To encode this exact behavior we could use a Heaviside (Heaviside(x) = $1_{R+}(x)$) but its derivative is equal to 0 except in 0 which makes it very hard to deal with artificial neural network training. Thus tanh is a function that has similar properties and has a better derivative.

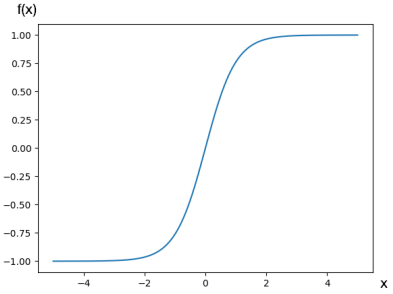



Figure 1: tanh function

**relu**

$$f_a(x) = relu(x) = max(x, 0) \tag{3}$$

Unlike tanh, relu (Fig 2) is unbounded which makes it a really good choice for more quantitative tasks (we will speak about this very difference later). Although it seems far from biological computations it is in fact another way to see biological neural networks. Biological neurons encode data in 0-1 representation (presence of spike) over time. This information isn't encoded in the value of the spike but rather in its frequency. When a neuron spikes very often it represents a high value where low-frequency spiking means almost 0. Relu could be seen as a frequency activation function making neurons speak over a short period of time (frequency) rather than at a specific timepoint like tanh does (spike or no spike). Therefore it still has a biological interpretation .

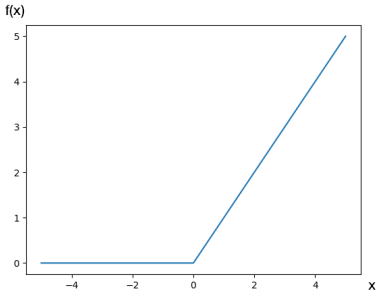



Figure 2: relu function

**softmax**

$$f_a(x) = softmax(x) = \frac{e^x}{\Sigma_{i \leq n} e^{x[i]}} \tag{4}$$

with x of length n and x[i] the i-th component of x.

Softmax (Fig 3) is more of a mathematical activation function as it transforms a vector in a probability distribution enforcing its existing structure (high values will result in high probabilities and low values will have almost 0 probabilities).

### 3.1.3 Simple Recurrent Neural Networks

In practice, many problems have a time dependency. Therefore it means the input will change over time and you want to integrate a sequence of inputs rather than only one. For example, if you work

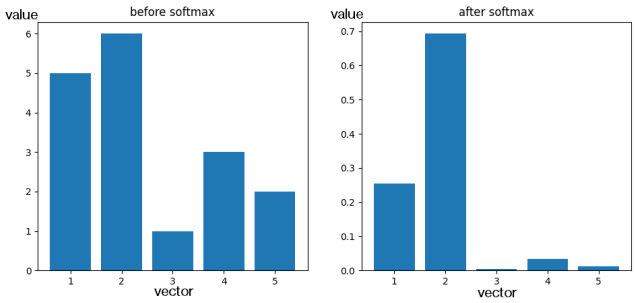



Figure 3: softmax function (left : vector of length 5 before softmax, right : vector after softmax

with videos and try to classify those which are sensible for kids you will start from the first frame from the video and go through each frame until the end. The network will see each image individually and must integrate the context to decide the sensitiveness of the video. To do that you can't use FC layers as they don't make it possible to integrate the context. FC layers are used for a 'feed forward' model which means the output of the network will only depend on the given input. Here we want it to be able to integrate a sequence of inputs and thus to remember what it did on previous calls. A way to do that would be to add the previous activation of the layer to its input next call.

Mathematically speaking :

$$v_o^n = f_a(W.v_i^n + W_s.v_o^{n-1} + B) \tag{5}$$

with n$\geq$ 1,$v_o^n$ the output activation at timepoint n ($v_o^0$=0), $v_i^n$ the n-th input and $W_s$ the recurrence matrix.

This quantity passed from a timestep to another is called *state*. In our Simple Recurrent Neural Network the output of a layer in determined by the input and the previous output (state) : $output = state_{n+1} = f(input, state_n)$. We will keep things simple here but it is important to know that most state of the art recurrent networks use more complex states. In these networks, because the state and the output can be different the equation becomes : $output, state_{n+1} = f(input, state_n)$.

This recurrence matrix represents connections between neurons of the same layer between two timesteps.

This way when calling the network on the second image it still has the first one in mind and knows how to integrate it with this new second image through the recurrence matrix.

When working on neural representation activations of a layer evolve in what we call a *latent space*. If we consider the last example about videos, the first frame of the video will activate some neurons of the recurrent layer, then the next image will change activations, etc... This will create a trajectory in the latent space which is often very interesting to understand. This trajectory will contain several points one for each frame of the scene (n-th point = layer activation at timepoint n).

For example, in Fig 4 if the video starts with a casual scene the trajectory might not move very much and stay in area A . However, if the scene right after is more violent it may move toward an area B. Showing another casual scene could not make it move to area A after because having one violent scene is enough for the movie to be classified as inappropriate for young children. We could see B as an absorbing state in Markov chains theory. It is this very dynamics and topology of latent space that we want to observe to have a good understanding of how neural networks behave and how those representations can cause errors.

Following the concept of the project, we can first see in this example that the latent space topology follows categories rather than stimuli (area A is safe / area B is unsafe). While creating this example I manage to put some sources of errors for the network as it takes several frames to go from A to B. However even very few violent frames could be enough for a movie to be classified as inappropriate. That is for example how we can understand the limits of neural networks by looking at their latent space and how they behave over it. A better representation would be a much smaller latent space from which we could go from A to B in one or two images and almost no movement when looking at casual scenes. Obviously, like humans, neural networks aren't perfect and such latent space dynamics might be hard to achieve in practice.

After this quick parenthesis about the aim of the project and ideas about how to observe neural behavior (that we will discuss later anyway) let's talk about how to train neural networks.

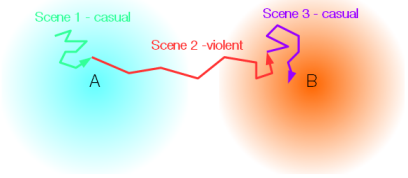



Figure 4: **Scheme of a latent space**. It is a representation of the neural activation evolution. In this scheme, two areas are representing the two modes of the network. Initial activation is in area A. Seeing casual scenes makes it stay in this area. However, when the system sees violent scenes, it switches to its second mode, represented by area B (which means a new neuron pattern is activated).

## 3.2 Training

Training an artificial neural network involves other rules than biological learning. A neural network is a continuous function from which we can compute a gradient and use it to learn.

First we need to define a loss function which will be minimized by a fully trained network. For example if our goal is to train a classifier of cat/dog images, from a batch of images $(x_i)_{i\leq n}$ and labels $(y_i)_{i\leq n} \in 0,1^n$ (0 = dog / 1 = cat) we can use the L2 loss :

$$L(w,x) = \frac{1}{n}\sum_{i\leq n}(y_i - nn_w(x_i))^2 \tag{6}$$

with $nn_w$ the neural network function with the set of weights $w$.

This quantity is positive and is equal to 0 when the neural network always predicts the right value on batch $x$.

From a batch x we can compute the gradient (vector of all derivative for each weight)

$$\frac{\partial L(x,w)}{\partial w} \tag{7}$$

and apply it to weights in order to modify the network to better fit data.

Here is the simple update rule

$$w_i := w_i - \lambda \frac{\partial L(x,w)}{\partial w_i} \tag{8}$$

with $w_i$ the i-th weight in the network, $\lambda$ >0 the *learning rate* (usually $10^{-3}$).

This process is called *gradient descent*. The intuition behind gradient descent is to take each weight individually and to observe how the loss behaves by increasing and decreasing the weight a little bit. This weight then evolves in the direction that decreases the loss.

Updating weights several times on the same batch will make the network perfectly fit data. Updating weights on a huge batch is often too costly in memory thus databases are split into many batches. The update rule is thus used on a random batch each step to have an average knowledge of the database.

In practice, people use more complex optimizers such as Adam which involves a momentum (physics analogy) in the learning rate to go down the gradient more efficiently.

## 3.3 Reinforcement Learning

Reinforcement Learning is a famous machine learning paradigm. It involves training an agent without labels $(y_i)_i$ as we don't always have them (think about chess when we know who wins but don't know the optimal move). The most well-known achievements of reinforcement learning probably are Chess and Go and more lately complex video games such as DOTA and Starcraft.

In RL we make the *agent* (usually a neural network) learn in an *environment* (chess game, ...).

Reinforcement learning aims at training the agent to get the maximal reward before the end of the run (end of the chess game for example).

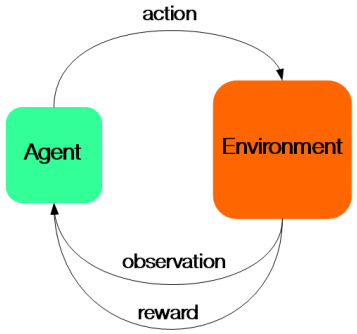



Figure 5: **RL system**. An agent and an environment compose a reinforcement learning system. The environment starts by sending an observation to the agent. Then the agent will react by choosing an action that will modify the state of the environment. After executing the action, the environment rewards the agent (positively or negatively) and sends a new observation until it reaches a final state (end of the game).

The agent sees observations from the environment (an Atari game for example) and chooses an action to take in this situation (buttons of the Atari controller). This action will return a reward (0 if the game continues or draws, 1 if the move wins, -1 if it loses).

The most common way for a neural network to compute actions is to propose probabilities for each possible action and then to sample from this distribution. For example in a 2d maze actions would be [up, left, right, down] (can be seen as the positions for the Atari joystick). The network will then return something like [0.1,0.4,0.3,0.2] and sample an action (move right for instance) which will be sent to the environment.

To make the agent learn we make it play the game for an arbitrary number of games (1 000 mazes game for example). It will constitute a batch of sequences of (observation,action,probability for chosen action,reward) to fit on : $(o_i, a_i, pa_i, r_i)$. We then need a loss function to train the agent to get more rewards (minimized by an optimal reward gathered through the game).

Most RL loss functions uses the *outcome* G for a game of length n (n moves) :

$$G(i_0) = \sum_{i=i_0}^{n} \gamma^{i-i_0} r_i \tag{9}$$

with $\gamma$ is the *discount factor* in [0,1[.

This quantity represents the reward gained after move n by focusing more on next few rewards.

### 3.3.1 The REINFORCE algorithm

The REINFORCE algorithm uses a simple loss :

$$L(w, x) = -\sum_{i \leq n} G(i) log(pa_i) \tag{10}$$

In last equation L depends on w (weights of the network) and x (inputs of the network) which are part of G and $(pa_i)_{i \leq n}$.

Even if the loss is simple its behavior far from trivial. If at step i G(i) is

- **positive** it means $a_i$ probably had a positive impact on the final outcome and minimizing the loss comes to maximising $log(pa_i)$ thus maximising $pa_i$ which will increase the probability to do the same action again.
- **negative**, $a_i$ probably had a negative impact on the final outcome and minimizing the loss means minimizing $log(pa_i)$ thus minimizing $pa_i$ which will decrease the probability to do the same action later.

Unlike supervised learning, we don't want to observe the loss during training as it won't decrease that much over time. We prefer plotting reward gained over time or accuracy to solve a task as it is far more visual and should increase as the training goes.

Even if we increase probabilities of effective actions it doesn't mean the agent will learn optimal actions. If the optimal action isn't ever tried in a situation its probability cannot increase and will never be chosen. For example, if the agent is in a situation where 3 actions are possible: A, B and C. Let's assume that for an arbitrary reason it outputs a probability 0 for C at the beginning of the training. This means it will have to choose between A and B for the first batch. It will learn which one of the two is the best during the learning phase (let's say it is A). Then it will choose A more often and slowly forget about B and keeping the probability for C at 0 knowing that A is better. However, it has never tried C and maybe C is far better than A.

That is why it is important to regularly try new actions even if they seem less optimal. There are 2 ways to do so :

- when sampling actions from probabilities given by the network we could sometimes (5% of the time for example) sample according to a uniform distribution instead).
- add an entropy term in the loss to try to always keep significant probabilities for each action.

These methods are only useful when we have more than 2 actions because with only 2 actions, increasing the probability for one will decrease the other and vice versa (as shown in the previous example the problem starts at 3 possible actions).

Training with REINFORCE means a first phase of playing to generate a batch of games $(o_i, a_i, pa_i, r_i)$, then to fit on this batch using the REINFORCE loss and gradient descent and do it over and over again. Each iteration is called an *episode*.

# 4 Our proposed experiment presentation

Our proposed experiment is a process designed by Julie Drevet and Valentin Wyart to enquire about how humans develop different neural representations while doing the same task by modifying the order of stimuli.

This experiment relies on a sequence of angles generated according to a category $c$. Two categories are possible : *orange* and *blue*. From a selected category a sequence of angles (arbitrary number of angles $n$) is generated by a Von Mises of parameter $\kappa$ (see Fig 6).

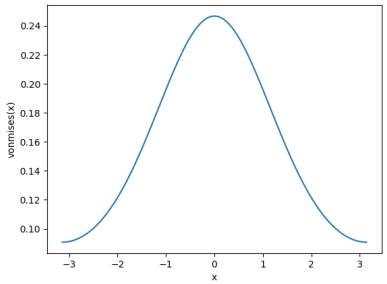



Figure 6: Von Mises distribution ($\mu$=0,$\kappa$=0.5)

Here is the generative process of a sequence of $(a_i)_n$ angles:

$$\begin{aligned} c &\sim U([|0,1|]) \qquad \text{0:orange 1:blue} \\ \forall i \leq n, a_i &\sim VonMises(c.\frac{\pi}{2}, \kappa) \end{aligned} \tag{11}$$

usually with $\kappa$=0.5 and n=12.

After seeing the sequence in a condition, the agent tries to guess the generative category.

Three conditions have been studied (Fig 7) :

1. CONTROL : angles are shown one by one along with the reference each time (if the sequence has 12 angles the agent will see 12 times the same reference)
2. PRE : the agent only sees the reference once **before** seeing all angles.
3. POST : the agent only sees the reference once **after** having seen all angles.

These conditions were made to test on humans as their memory and brain representations should differ from a condition to another. Using artificial agents, we could store the exact value of the reference and the given angles like in a computer making all three conditions equivalent.

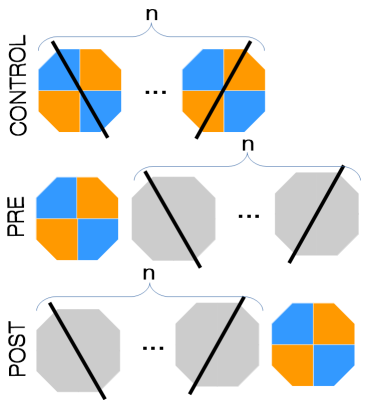



Figure 7: **Experiment Conditions**. In the CONTROL conditions, the network will see a new angle of the sequence and the reference at each timepoint. The PRE conditions only make it possible for the network to see the reference once **before** seeing any angle of the sequence. Lastly, the POST conditions only show the reference **after** seeing all angles of the sequence.

Human data collected by the team shows that CONTROL accuracy > POST accuracy > PRE accuracy (Fig 8). As we want to simulate human behavior we will try to have similar accuracy results. After that, we will check representations developed by the networks to compare them with magnetoencephalography (MEG) data when these are available (after the internship).

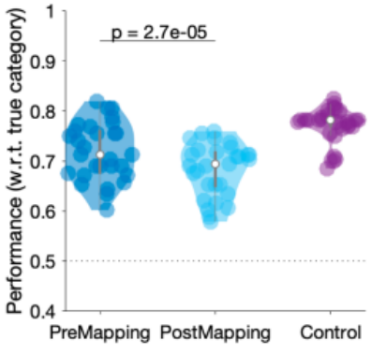



Figure 8: **Human performances**. This figure represents human data about classification accuracies in all three conditions. Here sequences shown have various lengths (4,8 or 12). Globally the CONTROL condition seems to be the easiest with almost a classification rate of 80% accuracy, followed by the PRE with approximately 72%, and lastly, the POST condition which seems to be the hardest with only 70%.

## 5 From standard networks ...

To solve this task we used several types of networks. Each idea for a network has to be implemented in all 3 conditions. This section is about the standard type of networks (the simplest). These very

simple networks could either have optimal behavior (which would still be interesting) or human-like results (which is ideally what we want to achieve in the end).

## 5.1 Networks presentation

Our goal is to test how neural networks behave in this kind of task. Then, we are going to compare their performances with human data.

That's why we need to introduce a way to encode angles and the reference for networks to interpret it correctly. Our data being angles from $[0,2\pi]$ we need an efficient way to encode the periodic aspect of angles for neural networks to be able to deal with them. Two ideas were proposed :

- Encode angles in a plane [cos(x),sin(x)]. This way the neural network should be able to work on this structure as it is quite intuitive for a structure that operates with matrix multiplications.
- Feed the network with actual angle representation ($\in[0,2\pi]$) but put cos activation functions in the network in order for it to be able to deal with this periodic aspect by itself.
- Feed angles with a representation similar to the biological visual neural system (LGN) with an array of neurons encoding an image with a specific resolution. Those neurons can be activated to draw angles on an input plan of neurons.

In the end, the first idea has been used and the second and third ones being too exotic haven't been tried.

Thus angles sequence, as well as the reference, will be represented by a [cos(x), sin(x)] array in artificial neural networks.

### 5.1.1 CONTROL network

The CONTROL condition needs to see angles and references at the same time. Therefore the input of the neural network will be [cos(angle$_i$), sin(angle$_i$), cos(ref), sin(ref)] and the output the probability for each category to be the one the sequence has been generated from (array of length 2).

We used a 50 neurons SimpleRNN with a tanh as recurrent layer and a 2 neurons FC layer with a softmax at the end to sample probabilities : Fig 9.

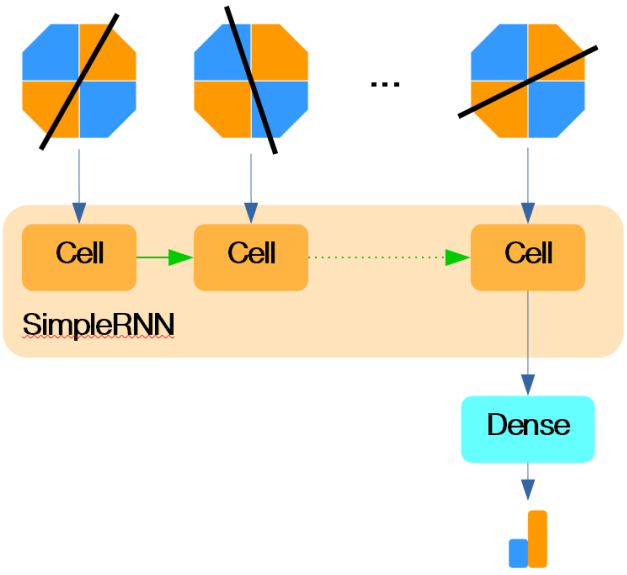



Figure 9: **CONTROL Network**. This network uses a SimpleRNN to integrate the angle sequence with each cell having four inputs ([cos(angle), sin(angle), cos(ref), sin(ref)]). The output of this network is then given to an FC layer which will compute probabilities for each category.

### 5.1.2 PRE network

The network for the PRE condition needs to see the reference before any angle which makes it more complex. We thought about 2 structures :

- As angles representation is a [cos(x), sin(x)] (a circle) we could fill with a [0,0] when no angle is given because it would not be seen as an angle (norm 0 instead of 1). We could then use the CONTROL network structure. However, there is an issue with this idea: the network might not understand how to interpret a [0,0] input. A neural network is a structure that transforms volumes at each layer. It starts from two circles and has to learn how to create new relevant volumes in order to estimate the generative category. Thus having a [0,0] point in the middle of the input representing something different than the volume itself might harden its understanding of the task.
- We could separate the handling of the reference and the sequence adding a layer before the recurrent part of the network in charge of computing the initial state for the recurrent network. This will make it possible to encode the reference in the state before seeing the sequence and then to estimate the right generative category while watching the sequence.

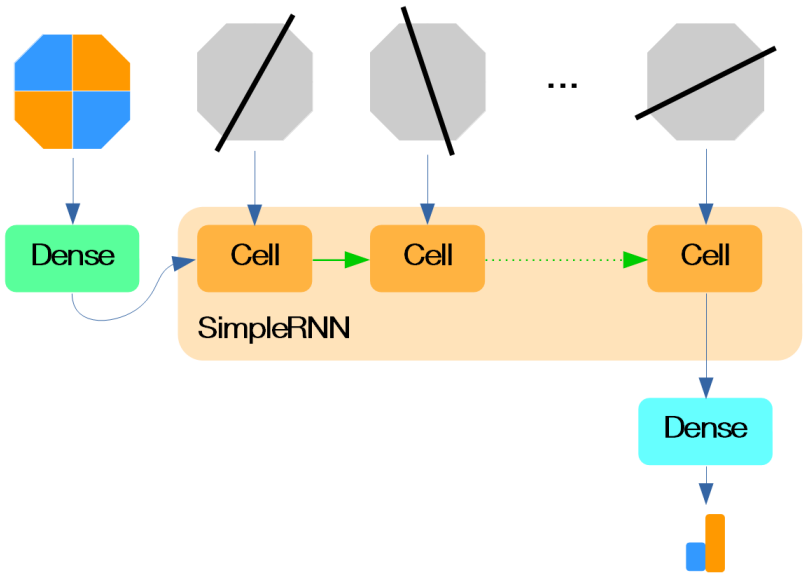



Figure 10: **PRE Network**. This version of the PRE network first uses an FC layer to deal with the reference and feed its encoding to a SimpleRNN. This SimpleRNN will integrate the angle sequence with each cell having two inputs ([cos(angle), sin(angle)]). The output of this network is in the end given to an FC layer which will compute probabilities for each category.

In the end we used the second structure as we didn't really know what would happen with the [0,0] representation issue.

We used a 50 neurons FC layer with a tanh to handle the reference, a 50 neurons SimpleRNN with a tanh as well for the sequence and a 2 neurons FC layer with a softmax to sample probabilities : 10.

### 5.1.3 POST network

The POST network has the same issues as the PRE network. Thus we used the same structure for the same reasons. However, instead of computing the initial state for the recurrent part, this additional part will get the state after the sequence has been encoded. This part will somehow deal with the reference before feeding it to the last layer which computes probabilities. Using this structure we let the network decide whether it's better to encode the reference accurately next to the sequence encoding or to directly estimate the generative category.

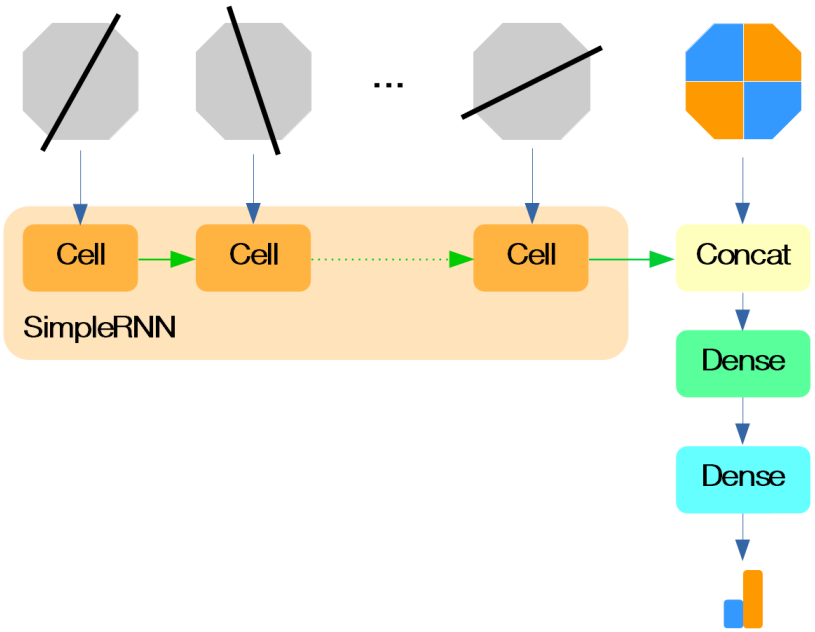



Figure 11: **POST Network**.The POST network uses a SimpleRNN to integrate the angle sequence with each cell having two inputs ([cos(angle), sin(angle)]). This encoding of the sequence concatenated with the reference is then fed to an FC layer which will encode the reference with the encoded the sequence. The output of this layer is finally given to an FC layer which will compute probabilities for each category.

We used 50 neurons SimpleRNN with a tanh for the sequence. We switched the 50 neurons FC tanh layer on the other side taking the vector output of the sequence and concatenating it with the reference and kept the 2 neurons FC layer with a softmax to sample probabilities at the end : 11.

## 5.2 Training

An agent gets a batch of 2000 inputs (sequences of 12 angles and 1 reference) using the appropriate representation depending on the condition (CONTROL / PRE / POST). The predicted category for each sequence is then sampled from probabilities given by the output of the network. We also keep the batch of real categories for learning.

Training consists of a simple REINFORCE algorithm with no regularization (not useful when only 2 outcomes are possible) using predicted output, predicted probability for this output, and the real generative category. The optimizer used is an Adam with a learning rate of $10^{-3}$.

An additional feature has been used as we don't want the network to learn to classify sequences of length 12 only but all length of the sequence. Thus by only using this pipeline, the recurrent behavior will very probably converge to a transitional regime for the first few angles (the network won't be able to classify if we stop the sequence at for example 5 angles) and the network will then converge to a decision. This is not what we want because it's a quite difficult behavior to analyze (in addition to the angle sequence dependence it also relies on time) and human experiments were done using multiple sequence lengths in the same batch.

Implementing a batch of inputs with different size is quite hard to do in practice thus we used batches of sequence length 12, ran the recurrent part of the network, extracted all 12 states (all transition states and not only the last one), and fed one taken randomly (uniform between 1 and 12) to the decision network. Thus the decision network will see the state corresponding to a sequence of a length between 1 and 12. We called this feature a Random Timestep Selector and is only used for training. For validation, it is replaced by the classic Last Timestep Selector.

## 5.3 Results

Using the Random Timestep Selector we have been able to make the three networks learn an almost optimal behavior (see Fig 12) (the optimal behavior being computed by a likelihood algorithm with a random timestep selector is 80%). Those accuracies have been computed by generating a new batch of data, computing this accuracy on the batch, and then feeding the batch to the network to learn (testing before training).

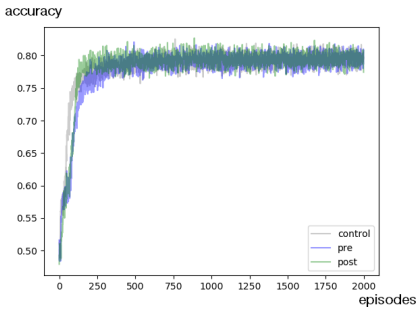



Figure 12: **Training accuracy**. All networks converged up to an accuracy of approximately 80%, which is the optimal accuracy for using a random timestep selector.

## 5.4 Analysis

We thus analyzed the network to understand better how it works in order to compare it with human data later.

Analyzing a neural network means understanding how it solves a problem and what could cause it to make mistakes. To do that we inspired ourselves of a paper [RS20]. They regularly use PCAs in latent spaces to observe its dynamics as well as regressions to verify whether it contains specific information.

### 5.4.1 POST network

The POST network is probably the simplest of all three. As a reminder, it gets the reference after seeing the full sequence thus the recurrent part only has the sequence as input.

To understand how it deals with the angle sequence we ran a PCA on the latent space. We only had 2 significant components (Fig 13).

Plotting those components shows they represent the mean angle of the sequence on a circle Fig 14. The input state being 0, seeing the first angle of the sequence will move the state to represent this angle. Then seeing the next angle will move in its direction etc ... In constant sequences, we see it only goes in the direction of the angle. Standard sequences try to compute the mean value between all angles. To investigate a bit further we have tried to understand the real impact of the input angle.

Another paper [DS21] proposed an interesting idea. In specific cases like ours when we can consider the recurrent neural network as a function $f(input, state) = new\_state$. This means we could see a recurrent neural network as a function that moves a point in the latent space according to an input. It could then be interesting to find fixed points. The main issue is that fixed points depend on the input. The paper proposes to remove the input by integrating over the input space but I don't think it's the best idea. I think inputs are as important as states and understanding the impact of the input on latent space dynamics is crucial.

Mathematically speaking a recurrent network is a function of 2 parameters : f(input,state) = output,new_state. However for our SimpleRNN output = new_state so we will simplify it : f(input,state) = new_state. We can then apply the previous idea of searching for fixed points. We can see the recurrent neural network as a function f which is a vector field moving states according to an input parameter. Figure 15 show this field for LEFT : input angle 0 and MIDDLE : input angle $\pi$. We see that states converge to a point representing this value (note that 0 and $\pi$ points are on opposite sides of the circle). The RIGHT figure show what happens with a [0,0] input ($(f(input, state) = new\_state = tanh(W_i.input + W_r.state + B)$). If input = [0,0] we remove the

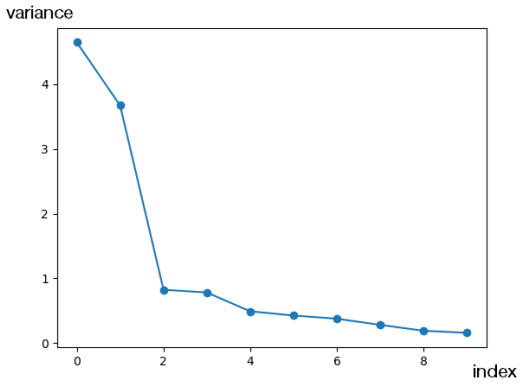



Figure 13: **Sorted PCA Variance**. This figure shows the ten first sorted variance values. This space clearly has two major components.

$W_i$ input matrix and see only the time impact on the system. By switching to circular coordinates we see that $\theta$ is only modified by the input where $r$ seems to be highly impacted by the time factor. Therefore $\theta$ seems to represent the mean value of the sequence and $r$ a time factor used to compute this mean value.

It's interesting to note that $\theta$ has little memory because by selecting 2 angles : $a_1$ and $a_2$ and feeding a sequence of high length of angles $a_1$ and after another of $a_2$ the states seems to forget $a_1$ 16. That's the first issue with the system: it has little memory and thus does mistakes that could be similar to what we know about humans.

About $r$ it seems like the clearer the sequence the higher $r$. Therefore it could be seen as a 'confidence' parameter that could be used in the decision network. Another approach would be to consider it as a mathematical way to perform an average operator on a circle. Usually, neural states don't change too much during a timestep. Thus if $r$ is high $\theta$ will be less likely to change which is a property of the average operator.

Mathematically speaking :

$$\begin{aligned} \bar{x}_{n+1} &= \frac{1}{n+1}\sum_{i}^{n+1} x_i \\ &= \frac{1}{n+1}(\sum_{i}^{n} x_i) + \frac{1}{n+1}x_{n+1} \\ &= \frac{n}{n+1}\bar{x}_n + \frac{1}{n+1}x_{n+1} \end{aligned} \tag{12}$$

If we compute the average of a long sequence of numbers the value won't change a lot by adding another number to the sequence. However, if we add an additional number to a short sequence its average value may change a lot. If $r$ is a manner for the network to compute an average its value should increase at each timestep as the sequence becomes longer and longer and that is what we observed in fig 15 RIGHT part.

Hence the recurrent network tries to estimate $\bar{x}_n$ at each step. This means the role of $r$ is to estimate $\frac{1}{n}$ in order to weights the estimate $\bar{x}_n$ and $x_{n+1}$.

Consequently $r$ has to decrease similarly to an inverse function. In Fig 15 RIGHT part and Fig 14 (a) and (b) we see that $r$ seems to grow less and less. This behavior is hard to learn from matrix multiplications and additions in Cartesian coordinates. Therefore this evolution pattern has to come from the tanh activation function.

This observation raises the question: what would happen if instead of a tanh we use a relu ? Could the network still learn this average operator? Will it still do the same mistakes?

Training with a relu ends with almost optimal behavior as well 17. However internal representations defer from tanh network. Tanh enforces a central representation due to the fact that tanh is bounded. Nevertheless, relu will try to have a sparser representation by spreading information in the positive direction.

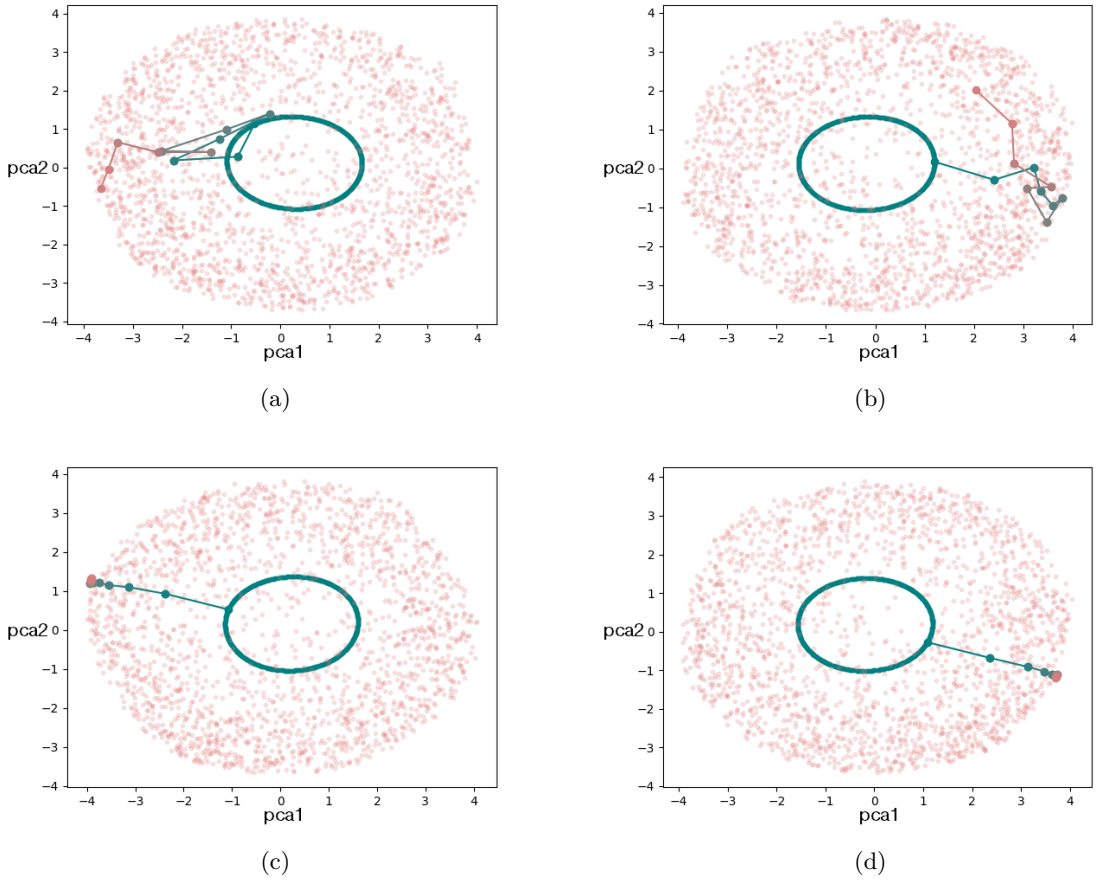



Figure 14: **Latent space of POST network after PCA**. Green spots (circle) are positions in the latent space after seeing only one angle on a full 2000 sequences batch. Red ones are after seeing 12 of them on the same batch. Four trajectories are shown : (a) and (b) are standard sequences (generated with kappa=0.5 $\rightarrow$ high variance) / (c) and (d) are constant sequences (generated with kappa=1000 $\rightarrow$ very low variance). The green circle shows that the network stores the first angle almost perfectly and that each following angle will move the system's activation toward the new angle position (a) and (b). (c) and (d) show constant sequences which are almost straight lines toward the angle position in the latent space (they always see the same angle).

In fact, 5 components are significant after running the PCA among which 3 have particularly high variance 18. Learned representations are quite similar to tanh representations as it's a cone-like shape ($r$,$\theta$,$h$) which looked from the top is very close to the circle of tanh networks ($r$,$\theta$). The main difference is that relu has a better understanding of time through $h$ parameter which increases the maximum value for $r$, the weight parameter to compute the mean angle value.

In the tanh network, $r$ was bounded which makes the network unable to accurately compute the average value of long sequences as it limits its memory. Here the unbounded activation function makes it possible to more accurately represent the time axis (unbounded).

In the tanh network, the time aspect of the computation was put on $r$ which is a bounded variable. During training, the network has been used to see sequences of short size (1 to 12 angles). It has thus learned to split this interval in order to represent correctly sequences of this length. However, giving longer sequences shows the limits of the tanh representation 16.

Nonetheless, with a relu it's still possible to update the count even for longer sequences 18 as $h$ is unbounded and will increase the maximum value for $r$ to compute the mean angle more accurately. Nevertheless, in practice, the accuracy difference between relu and tanh network on very long sequences (length 120) isn't very clear.

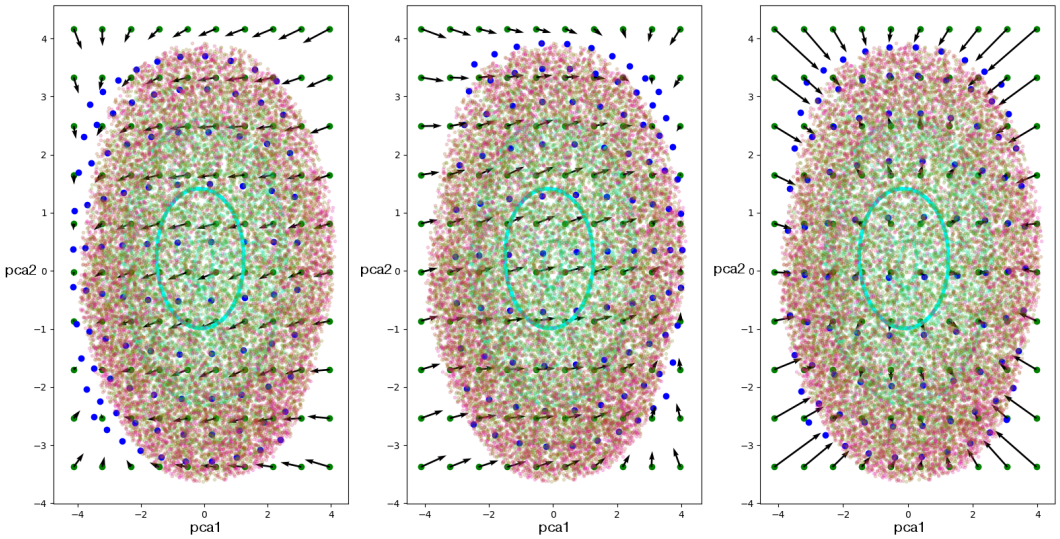



Figure 15: **Latent space dynamics**. Gradient from green to purple shows positions in the latent space (1 angle = green / 12 angles = red). Additionally to this background green grid shows points in the latent space and their transformation (in blue) after one step by the recurrent layer. The vector field represents this transformation. LEFT : input angle 0 / MIDDLE : input angle pi / RIGHT : input angle empty. Here we can additionally see the vector field moving the system's activation toward the angle position in the latent space (LEFT and MIDDLE). The RIGHT figure shows this field without any input which means the effect of the recurrence matrix as well as the activation function in the latent space.

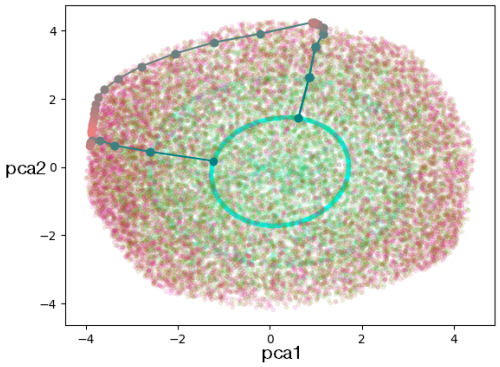



Figure 16: **Latent space deviation**. The green to red gradient shows the system's as more and more angles are seen. The Turquoise points are the system's activation after having seen only one angle. Three trajectories are plot : a sequence with 8 angles of 0, a sequence with 8 angles of $\frac{pi}{2}$ and a sequence with 8 angles of 0 and then 16 angles of $\frac{pi}{2}$. With this plot we can see an error cause with this neural network: with longer sequences, it tends to forget the beginning of the sequence.

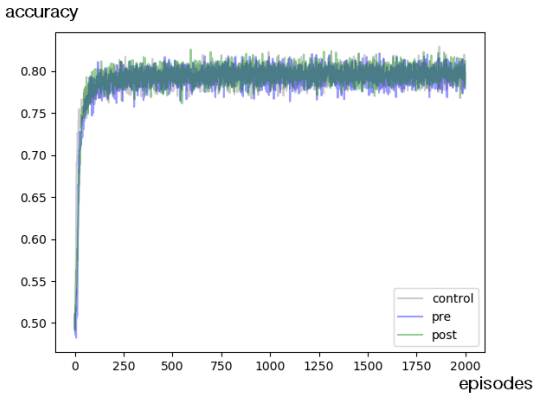
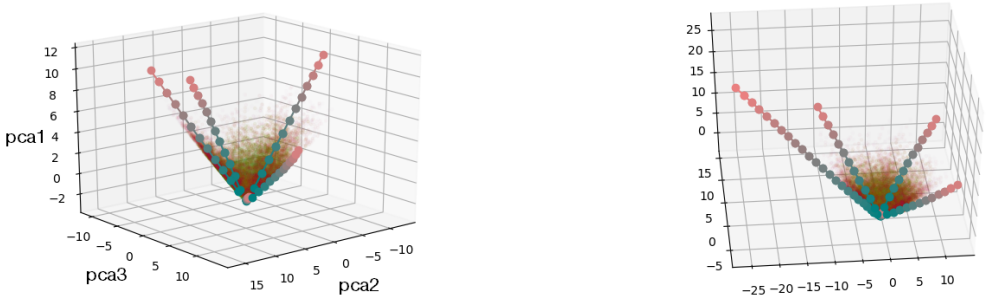



Figure 17: **Training curve for relu training**.All accuracies are optimal (80%) and close to the tanh training.




(a) (b)

Figure 18: **Latent space of relu POST network after PCA**. The green to red gradient shows the system's as more and more angles are seen. The Turquoise points are the system's activation after having seen only one angle. Four trajectories are shown on each figure : $\frac{2i\pi}{4}$ for i in [0,3]. Fig (a) shows trajectories for constant sequences of length 12 of each angle. Whereas (b) shows the same figure but with one of the sequences of size 24 (the left one). The main difference between tanh and relu networks is the number of significant components. This can be seen in the latent space as tanh POST network was a disk when relu latent space is more like a cone. This cone still has similar properties to the disk as the height seems to be a time line (a) which continues even on very long sequences (b) .

### 5.4.2 PRE network

The pre-network sees the reference before any angle and then sees the full sequence. It thus has to memorize the reference angle and ideally compute the best category on the fly.

Looking at the latent space with a PCA we found out that there is one major component and many minors 19.

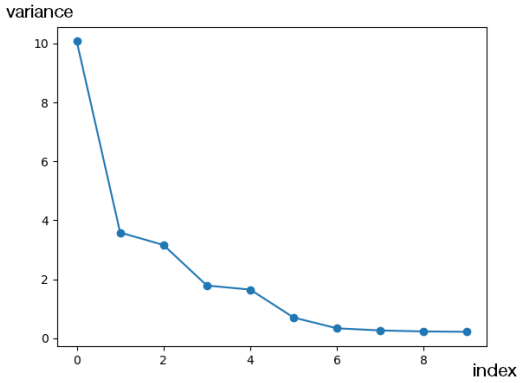



Figure 19: **Sorted PCA Variance**. We see there is one major component, but one component isn't enough to truly understand the dynamics of a neural network, so we have investigated the first three. However, looking at axes 2 and 3 makes axes 4 and 5 significant too as they almost have the same variance.

This latent space structure shows 8 turquoise circles Fig 20 (b) one for each possible value for the reference angle. Circles represent the first and unique angle seen at this stage of the computation. Blue and orange dots show that streaks axes make it possible for the network to guess the right category Fig 20 (c). Therefore points are sent to one of the 8 streaks Fig 20 (a) and will move along this line in the direction of the generative category dynamically while seeing the sequence of angles.

The fact that the main component has a huge variance (Fig 19) and represents the category (20 (c)) means this network computes the category on the fly. It means it almost only represents the category in the latent space instead of the stimuli like the POST network does.

### 5.4.3 CONTROL network

The control network is much more complicated than both previous networks. It sees the angle sequence as well as the reference at each step.

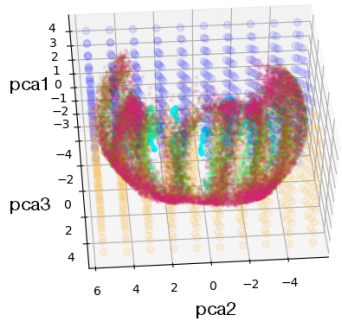
The PCA found 5 significant components which were quite hard to deal with as it is not easy to visualize (Fig 21).

We thought the major axis might probably be the predicted generative category as it could be dynamically computed while the sequence is processed by the network. The output of the network being this quantity the 4 other components need to be relevant to complete the approximation stored in the first axis. Thus it might be the mean axis as well as the reference (each stored with a [cos, sin] representation making 4 axes at the end).

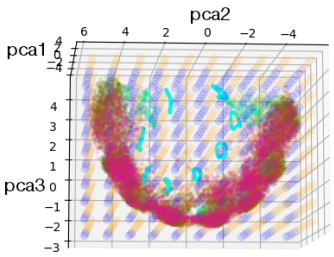
Plotting the 3 major components didn't show anything very relevant as 5 guessed axis isn't the basis found by the PCA but probably a slight rotation of it.

We had to verify this hypothesis. After the recurrent latent space, there is only one FC layer to sample probabilities which is almost a linear layer. Therefore we should be able to decode those axes almost linearly. Starting from the latent space we tried to linearly decode the reference from the system's activation at the end of each sequence. This decoding worked well with a Pearson determination coefficient of 0.95. We did the same for the generative category as well as the mean angle but determination coefficients were lower (resp. 0.56 and 0.68). This difference in coefficient might be linked to computation mistakes of the network.

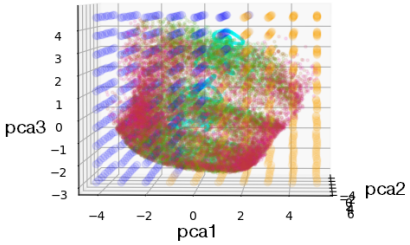
The reference is clearly stored linearly in the network which means we could remove it from the PCA 5 dimensional space and plot the 3 last dimensions to understand how it works. We thus computed the residue of the latent space by the reference and ran the PCA once again resulting in 3 significant




(a) view 1




(b) view 2




(c) view 3

Figure 20: **PRE network latent space**. The green to Red gradient represents points obtained from seeing few angles to a lot of angles. Turquoise points are positions in the latent space after seeing only one angle. Blue points are those decoded in a blue generative category (considered as very likely blue by the network) and orange ones are the ones for the orange category. It is important to know when looking at this latent space that we are looking at three of the first components even if five are significant. The pca1 axis is the most important by far and seems to be the category axis (see (c)). The two other axes shows 8 streaks which could be the 8 different values for the reference (see (a)). Turquoise circles show a complex encoding of the first angle relative to the reference. This complex representation is probably linked to the 3D representation of a 5D space.

components : Fig 22. Plotting those components shows a cylinder containing expected quantities : category axis and mean angles estimation 23.

The pca after reference removal shows 3 components (21) a major one and 2 minors. The major stands for the category (23) while the others for the mean angle. In the full pca (22) we see there a 3 groups : one major and 2 couples of minors. It is very likely for the major to be the category and for each couple of minors to be the reference and the mean angle. This means the CONTROL network computes the category on the fly as this axis has a huge variance.Nonetheless, unlike the PRE network, the reference and the mean value of the sequence also have significant variance. Therefore this network doesn't only compute on the fly but also averages angles like the POST network does. It has both category and stimuli representations.

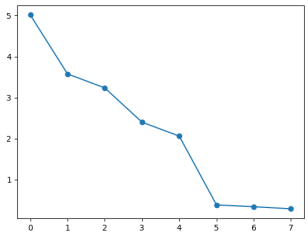


Figure 21: **Sorted PCA Variances**. There are 5 clear major components.

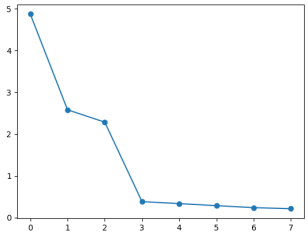


Figure 22: **Sorted PCA Variances after reference removal**. This time only 3 major components are shown.

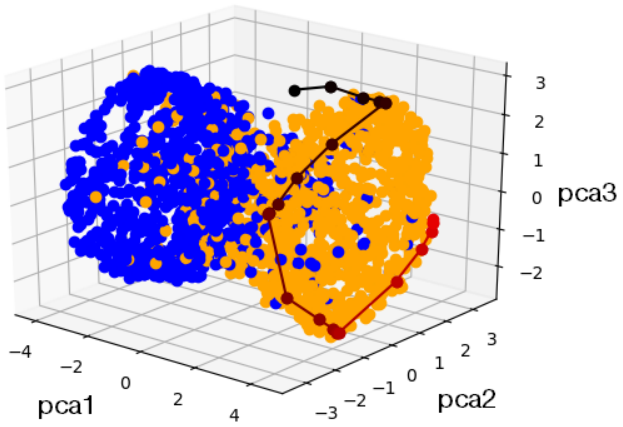



Figure 23: **CONTROL network latent space**. We have been able to reduce the dimension of the latent space by removing the reference. Nonetheless, it is important to remember that the reference component (which is probably a circle) is still part of the full 5D latent space and that we are only looking at a simpler representation of it. For this network we have used a different color code as blue points are points from a sequence generated from color blue and orange points from the orange color. A custom trajectory is plotted with a blue to red gradient. This sequences using a reference of 0 starts with an angle of 0 for few steps. Then switches to a $\frac{\pi}{2}$ angle for few steps, switches to a $\pi$ angles and finally to a $\frac{3\pi}{2}$ angle. The shape of the latent space seems to be a cylinder with a major axis (pca1) about classification and two other axes about computing the mean angle. It still has the same memories issues than the POST network as the mean angle representation seems to forget first angles (see trajectory).

### 5.5 Conclusion on standard networks

In the end, every training worked well and trained an almost optimal network. The PRE network developed on the fly category computation, the POST network averages, and the CONTROL does both. However, these networks don't make similar mistakes to humans. Therefore it is interesting to try to modify them a little bit to have more human-like results and to compare their representations to human MEG data and optimal network data.

## 6 ... to more human like networks

Modifying the network means changing its architecture to change learned representations for them to behave more like human data. This means we could change the RNN structure (use more complex RNN such as LSTMs, GRU, ...) but according to a paper [NM19] changing units doesn't affect representations that much. In fact, using SimpleRNN or LSTMs seem to very often develop the same understanding of a problem but will only slightly change the behavior over this structure (if the internal structure is a cube, using a LSTM instead of a SimpleRNN might harden a little bit corners for examples but it will still stay a cube).

Therefore we need to be more inventive to change this optimal behavior into another one that will do the same mistakes as humans.

### 6.1 Separated Networks

We first thought about links between the reference and angles from the sequence. In PRE for example the networks see the reference, then all angles and need to store them both in the same latent space. It could be hard for neurons to perform such a computation with high accuracy as the reference would be seen very early in the process and forgotten then. We thought maybe in the brain this reference would be processed by a specific group of neurons different from the one computing the angles sequence. Therefore this group of neurons could regularly input on the other group to compute the generative category 24 making the PRE system close to the CONTROL condition.

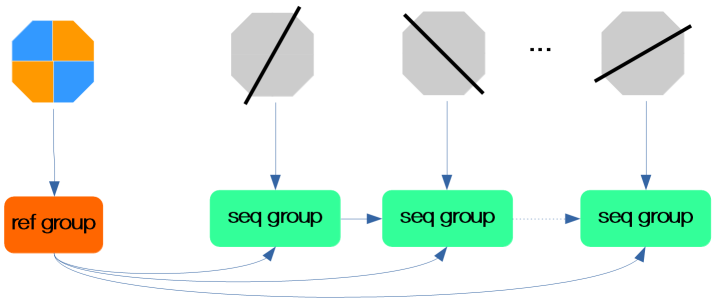



Figure 24: **Separated RNN principle**. The idea behind Separated networks is to have a group of network which would input an encoding to the others like if it was part of the input. This idea comes from the PRE condition as the network only sees the reference once before the sequence. This way it could be like in the CONTROL condition. This behavior happens in biological neural networks and could be interesting to try here.

However, it might be useful to have feedback from the sequence neuron group to the reference group. Thus to implement this idea we will divide the layer in 2 groups : *ref* and *seq*. The *ref* group will only get inputs from the reference when the *seq* group will only see angles from the sequence. This will require a structure similar to the CONTROL condition 9 for all 3 conditions with separated networks. When data is missing we will put a $[0, 0]$ instead (we have already talked about this trick earlier and its possible consequences on volume transformations but we wanted to try it anyway). Hence all 3 networks will have the CONTROL network shape with $[0, 0]$ where data are missing. Inputs for PRE and POST will then be like in Fig 25. The separated layer is in Fig 26 dividing the layer into two parts: one which will only get inputs from the reference (and zeros if the reference cannot be seen at the time point) and the other only seeing the sequence (or zeros).

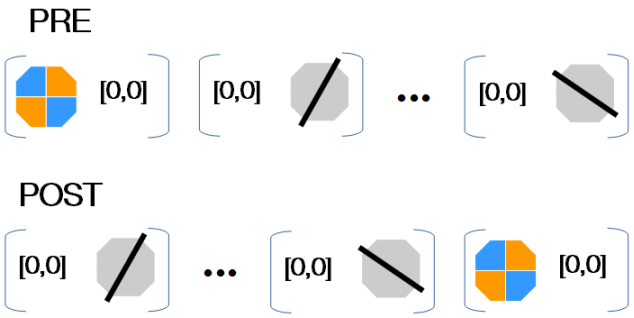



Figure 25: **Separated RNN scheme**. For these networks we will keep the standard CONTROL architecture but using the idea of puting a [0,0] where data is missing. For the Separated CONTROL network nothing will change as the input is always a full 4D input ([cos(angle), sin(angle), cos(ref), sin(ref)] but for PRE and POST networks it will change. For the PRE network the 4D input starts with the reference only (putting a [0,0] for the angle) and then angles with a [0,0] instead of the reference. The POST network input will be using the same scheme but with the reference at the end.

We have only used separated layers for the recurrent latent space. Our input being [cos(angle), sin(angle), cos(ref), sin(ref)] it makes the input matrix $W$ a block matrix

$$\begin{pmatrix} 0 & W_1 \\ W_2 & 0 \end{pmatrix} \tag{13}$$

with group 1 seeing the ref while group 2 sees the sequence. Nonetheless it doesn't change the recurrence matrix $W_s$ as we still want feedbacks to exist between both our groups.

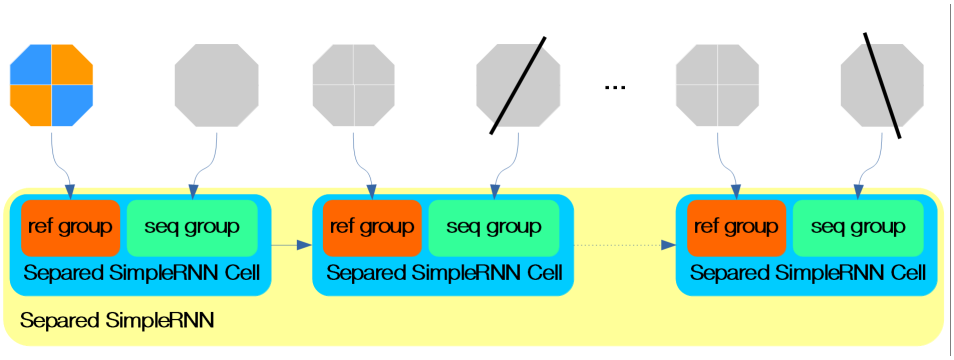



Figure 26: **Separated RNN scheme**. A separated RNN is a RNN which implements the separated principle explained above. The layer will be split in two and each layer will see a different part of the input: here the first group will see the reference while the other one will see the angle. The recurrent behavior of the layer is kept intact as the two groups can communicate.

Training separated networks brings results slightly different from standard networks as CONTROL $\approx$ POST $>$ PRE (which is not what we have observed with human data) 27.

## 6.2 Noisy Networks

Human neural computations do mistakes. These mistakes can be modeled by adding noise after the activation function [FW20]. Noisy layers functions will then be

$$v_o \sim N(l_f(v_i), std) \tag{14}$$

with $l_f$ the previous layer function and std a fixed parameter.

We then tried to train the same networks with a noisy latent space by switching the SimpleRNN layer to a noisy SimpleRNN layer with different values for *std*.

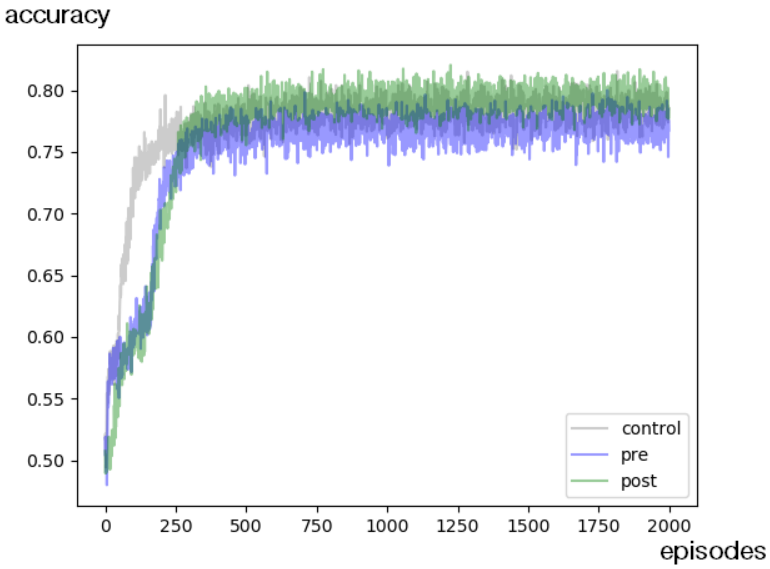



Figure 27: **Training of a separated network**. Compared to previous training the POST network seems to be better than CONTROL which is better than PRE. This isn't really what we want as even if the fact that PRE is harder than the others it is not exactly human behavior. However, it is still encouraging and improving it could be a good idea to reach human performances.

Results are very different than with standard networks: Fig 29. In fact, we see that POST > CONTROL > PRE. This order is weird because for humans CONTROL > PRE > POST. The fact that POST > CONTROL is particularly bizarre as POST only sees the reference at the end compared to CONTROL which sees it every time during the sequence.

These results can be explained by looking at the topology of the network. Using a noisy recurrent latent space for the PRE network the reference will be very noisy as it will accumulate 12 times the noise if the sequence is of length 12. It explains terrible results obtained with PRE conditions as in the end the reference is very blurry (that's why we have 50% accuracy with high *std*). CONTROL also accumulates noise on the reference while POST doesn't because the reference is seen after the recurrent latent space. Even if we put noise on every layer it won't change much as in POST the reference still accumulates less noise than in PRE.

This observation raises a question: How is the noise distributed in human networks? In practice, we observe that in human PRE seems to be easier than POST which means the reference has taken far less noise than in our system. It means the reference and the sequences are stored in different places where the reference doesn't accumulate too much noise during the sequence. We could thus use our separated networks and only putting noise in the group of neurons that sees something (different from zeros).

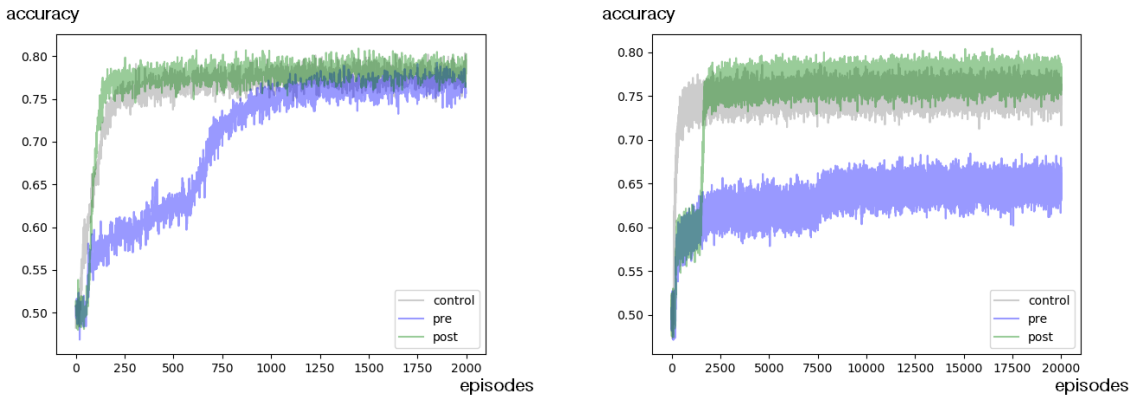
## 6.3 Noisy Separated Networks

Noisy separated networks are separated networks with a specific noisy SimpleRNN layer. This layer only applies noise if the input of the group isn't [0,0]. This means neurons only accumulate noise when they are directly activated by the input. Their recurrent behavior stays the same as in deterministic separated networks.

Training curves clearly show the human difficulty order: CONTROL > PRE > POST which is very encouraging (Fig 29). The std=0.5 training has been run 10 times from 2000 episodes to 7000 episodes and it always gave the same type of figure.

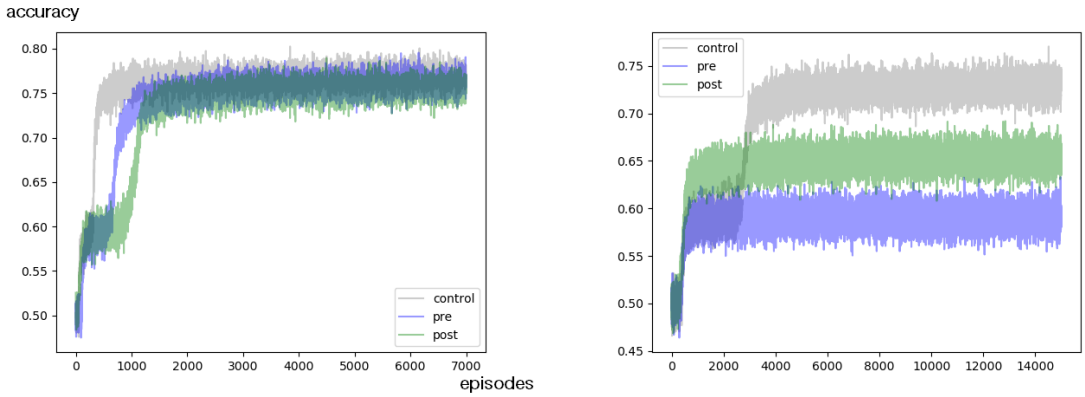
Their accuracies were quite close to humans ones so we have investigated their representations.

Those networks have similar shapes to standard networks but with some specific changes which could explain accuracy differences.




(a) std=0.5 (b) std=1.0

Figure 28: **Training of noisy networks**. Noisy training is longer than standard trainings which is why we trained on more episodes especially for high std. Noisy networks don't reach optimal behavior and POST seems to be more effective than CONTROL which is more effective than PRE. These differences seems to increase as the standard deviation increases. This order is the same than separated networks so it is still not what we wanted.

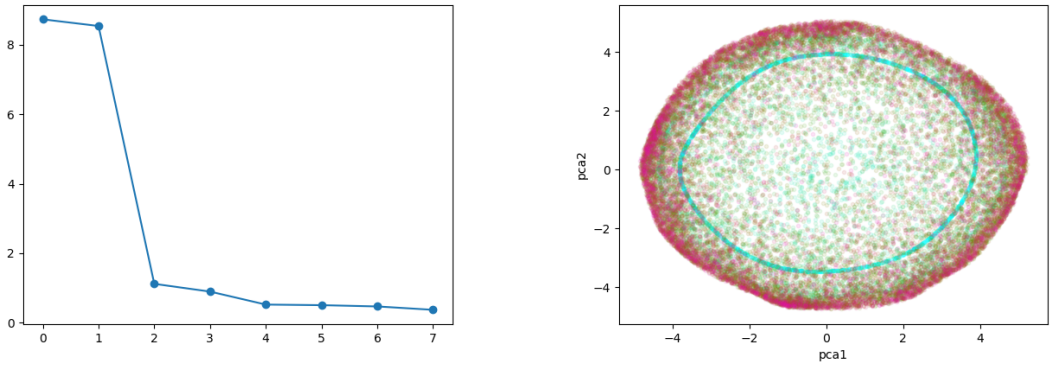



(a) std=0.5 (b) std=1.0

Figure 29: **Training of noisy separated networks**. Noisy and separated training seems to have similar results to humans as CONTROL accuracy > PRE accuracy > POST accuracy. The higher the std the higher the difference which means that somewhere between 0.5 and 1.0 there is the right std for almost human performances. These results are very promising which is why we have investigated how these features for neural networks bring human like performances.

## 6.4 The POST separated & noisy network (std=0.5)

The POST networks still develop a 2D latent space with the same structure 30. However, there is a major difference in the position of the initial angle circle. This circle is much more at the edge of the disk. This is probably due to the noise as the goal of this latent space is to compute the average angle of the sequence and using the center of the disk isn't reliable as $\theta$ would be too noisy. It is thus an attempt of the network to control the noise in this computation. We also see that red shapes are a lot clearer than in standard POST network latent space. This means almost all computations end at the edge of the structure.




(a) Sorted PCA Variances

(b) PCA of the Latent space.

Figure 30: **Latent Space of the separated & noisy POST network**. The PCA shows 2 major components (like in the standard network - see (a)). The green to red gradient shows system's activation as more and more angles are seen. Turquoise points are system's activation after having seen only one angle. The latent space has the same shape (see (b)) but the turquoise circle is a lot larger.

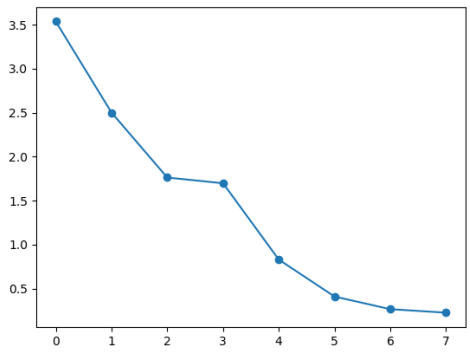
Not using the center of the disk makes it difficult to accurately compute an average which explains why its performances are worse than its standard version. Its sources of errors are that it has difficulties integrating very noisy inputs (if all angles are very far from one another it will have extra difficulties estimating the average (somehow it starts making errors like humans due to noisy layers).

## 6.5 PRE

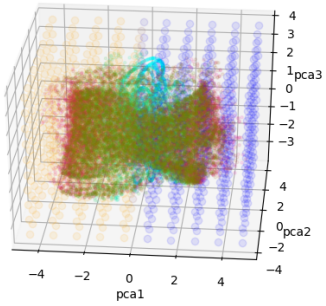
It's hard to compare previous figures of PRE network as they were the 3 first components of a 5D space which means they might differ a lot. Thus we have used the trick used on the CONTROL network. We tried removing the 2 components coming from the reference from a space that seems like a 5D space in order to make it a 3D space. However, with the standard network, a fourth component seems to appear 31 after reference removal. We thought it might be a time component (as the $r$ parameter in POST network) but Linear Regression doesn't work (determination coefficient of 0). We thus investigated three first components to see if something weird appears and we saw that the reference still appears even after linear removal. Multiple rows are visible (even if they are quite vague which makes it hard to see the 8 of them) as well as several very distinct initial circles (turquoise) (Fig: 31. It seems like the reference is the foundation of this latent space as it has been carried on multiple times across the layers (and therefore has stacked non-linearities).

In the separated & noisy network we see a representation very close to the CONTROL network (Fig : 32). Moreover PCA variances are close to the standard CONTROL network which means it doesn't only compute on the fly anymore but also averages. Our idea of separating both parts in order to create a layer that will input several times an encoding of the reference seems to have worked. This huge difference could explain why the PRE network is worse than the CONTROl network for separated noisy networks as it has to encode the reference once and then feed it to the sequence neurons. However, this encoding is noisy whereas in the CONTROL condition the network will be able to see the reference multiple times thus having a more accurate representation of it. Interestingly

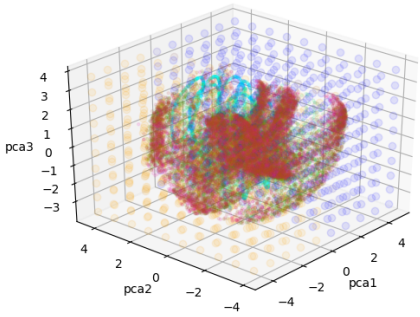
this network changed from category-based only to an hybrid representation with this new architecture. It will be very interesting to compare its representations with humans.



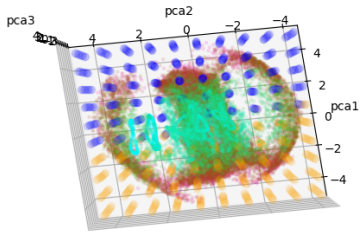
(a) Sorted PCA Variance after reference removal




(b) PCA of the 3 first dimensions of the 4D Latent Space (after reference removal) (view 1)




(c) PCA of the 3 first dimensions of the 4D Latent Space (after reference removal) (view 2)



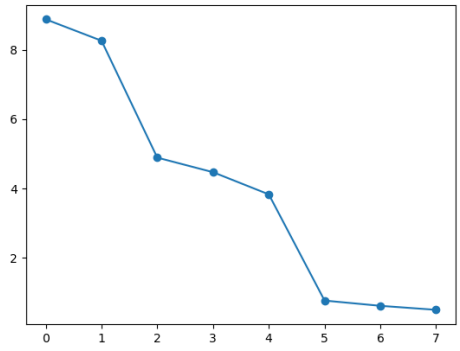

(d) PCA of the 3 first dimensions of the 4D Latent Space (after reference removal) (view 3)

Figure 31: **Latent Space of the standard PRE network after reference removal**. The PCA shows several major components (probably 4). The green to red gradient shows the system's as more and more angles are seen. The Turquoise points are the system's activation after having seen only one angle. It is hard to visualize a 4D latent space so we choose to investigate the first 3 components. This latent space is quite complex as it is a 3D version of a 4D latent space which explains why turquoise shapes (see (b), (c) and (d)) are unclear. This latent space seems to be a mix of standard CONTROl network with reference removal as (a) shows the same kind of structure but (c) and (d) seems to have streaks like the standard PRE network without reference removal. This is due to the fact that the reference is still in this space even after linear removal which means it isn't stored perfectly linearly in the full latent space.

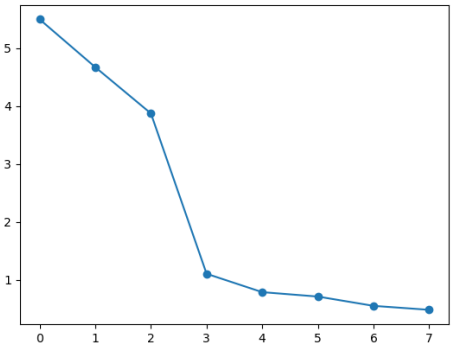
## 6.6 CONTROL

In the CONTROL network, looking at the latent space after having removed the reference (to go from a 5D latent space to a 3D space) we observe similar behavior to POST network especially about red edges (Fig: 33. It also seems to have a larger initial circle (see turquoise circles). It seems like it has several circles as the PRE network but they are less distinct from one another which means the reference still has gone through all these layers (and stacked non-linearities) but the fact that seeing it at each timestep made it possible to still keep an almost linear representation. This network keeps both category-based and stimuli-based representations even with the new architecture.

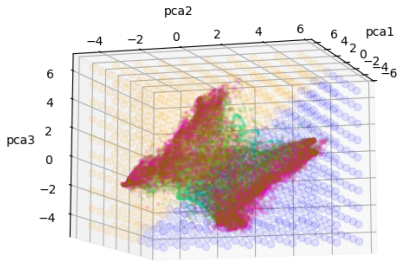
These changes don't affect its accuracy very much as contrary to the POST condition it doesn't



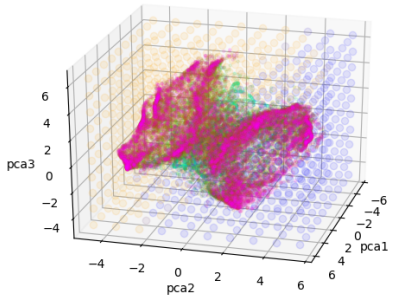

(a) Sorted PCA Variances before reference removal




(b) Sorted PCA Variances after reference removal



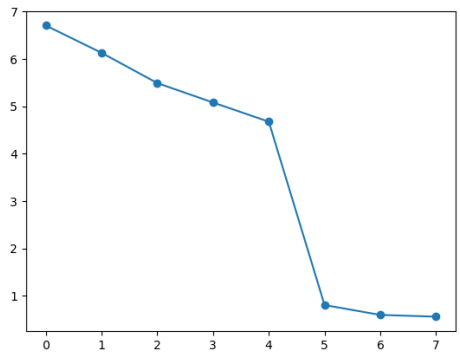

(c) Latent space after reference removal (view 1)



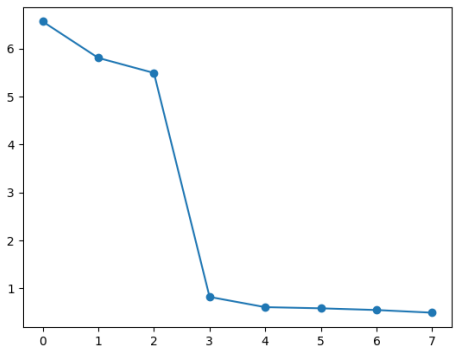

(d) Latent space after reference removal (view 2)

Figure 32: **Latent Space of the separated & noisy PRE network**. The PCA shows 5 major components (see (a)) reduced to 3 after reference removal (see (b)). The green to red gradient shows the system's as more and more angles are seen. The Turquoise points are the system's activation after having seen only one angle. activation This means compared to standard PRE network with reference removal this time the reference is linearly stored as no clear streak appear in the latent space.

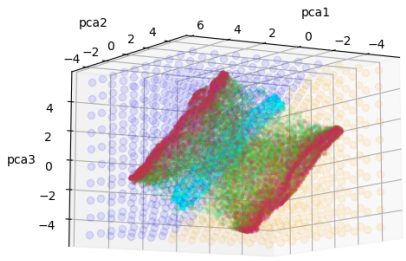
need to compute an accurate mean angle as it sees the reference during the full sequence. Therefore having a good estimate of this mean angle isn't as important as in the previous case which explains why its accuracy is less affected than the POST network.



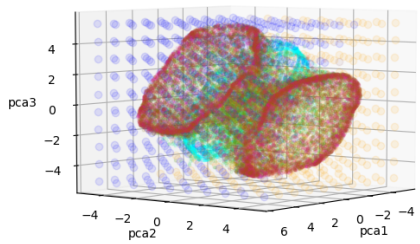
(a) Sorted PCA Variances before reference removal



(b) Sorted PCA Variance after reference removal




(c) Latent space after reference removal (view 1)




(d) Latent space after reference removal (view 2)

Figure 33: **Latent Space of the separated & noisy CONTROL network**.The PCA shows 5 major components (see (a)) reduced to 3 after reference removal (see (b)). The green to red gradient shows the system's as more and more angles are seen. The Turquoise points are the system's activation after having seen only one angle. This latent space is very close to the standard CONTROL latent space but with features from separated & noisy POST network like the large turquoise circle.

# 7 Conclusion

In the end we have trained recurrent neural networks to solve the categorization task performed by human subjects and studied in detail the representations used by the neural networks to solve each condition of the task. In the absence of internal noise in the networks, the recurrent neural networks were capable of solving each condition near optimally - better than tested subjects - and relied on different formats of representations to solve each condition (category-based for PRE, stimuli-based for POST and both for CONTROL).

After studying the neural networks trained in each condition, we have added internal noise to each neural network to reproduce the type of internal variability observed during visual integration in human subjects. We found that an architecture where the representation of the categories is not corrupted by noise and only the integration of visual stimuli is corrupted by noise is capable of reproducing the differences in human performance between the conditions. This new architectures changed representations of the PRE network as it switched from category-based only to hybrid representations.

These two important observations will be used in the later parts of the project, where the goal will be to analyze the recorded brain data in MEG from participants tested in this task using analyses modified from the ones performed in this project. The goal will be to test whether the neural representations used by human participants resemble the ones used by neural networks to solve the task. The link between noisy neural networks and human performance will also be strengthened in later parts of the project to investigate further the degree of resemble between artificial and human agents in this task.

# 8 Experience

I really enjoyed working on this project as it is a field of research I wanted to work on for a while. I did many projects by myself about simulating biological behavior using artificial neural networks and this internship made me learn a lot about this idea.

It was the first time I have seen what is inside a neural network and how it works which was a real plus for my understanding of neural networks. Starting from when to use different activation functions and their impact on the neural network behavior to the effects of using noise to force the network to make computation mistakes to have a more biological behavior. I think I have learned a lot during this internship in the Deep Learning field as well as its interface with neurosciences and mathematics.

After this internship, I think I will focus more and more on this direction for a future thesis.

# References


[DJ12] Rust NC DiCarlo JJ, Zoccolan D. How does the brain solve visual object recognition? *Neuron*, 2012.

[DK16] Devauchelle Drugowitsch, Wyart and Koechlin. Computational precision of mental inference as critical source of human choice suboptimality. *Neuron*, 2016.

[DS21] Omri Barak David Sussillo. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *MIT Press Journal*, 2021.

[FW20] Charles Findling and Valentin Wyart. Computation noise promotes cognitive resilience to adverse conditions during decision-making. *bioRxiv*, 2020.

[GS07] Joshua I. Gold and Michael N. Shadlen. The neural basis of decision making. *Annual Reviews in Neuroscience*, 2007.

[MN14] Shenoy Mante, Sussillo and Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 2014.

[NM19] Matthew D. Golub Surya Ganguli David Sussillo Niru Maheswaranathan, Alex H. Williams. Universality and individuality in neural dynamics across large populations of recurrent networks. *NeurIPS*, 2019.

[Rat04] Smith P. L Ratcliff, R. A comparison of sequential sampling models for two-choice reaction time. *Psychological Review*, 2004.

[RS20] Leenoy Meshulam International Brain Laboratory Ila Rani Fiete Rylan Schaeffer, Mikail Khona. Reverse-engineering recurrent neural network solutions to a hierarchical inference task for mice. *BioRXiv*, 2020.

[WW21] Lee Drugowitsch Weiss, Chambon and Wyart. Interacting with volatile environments stabilizes hidden-state inference and its brain signatures. *Nature Communications*, 2021.