Spatial Representations and Memory in Deep Reinforcement Learning

Nicolas Yax
ENS Paris Saclay

Paper : [3].
Code

Abstract

Reinforcement Learning and Deep Learning have been widely used in the last few years to model human behaviour and neural representations in various contexts. Combining both made it possible to go even further in investigating the links between behaviour and neural representations. Inspired by a paper from DeepMind reviewing state of the art challenges in Deep Reinforcement Learning for neuroscientific investigations I have studied the development of spatial representation in a maze solving task and compared different models of memory leveraging the importance and the role of computational units in representations developed during training.

1. Introduction

Deep Reinforcement Learning makes it possible to train neural networks on complex tasks such as Chess, Robot motion planning or even Atari Games. Many of those fields were already mastered by humans and are investigated by neuroscientists to understand the elementary phenomenons of cognition. Deep Learning from an other point of view is already used to model how humans learn to classify images, videos, ... [5] and Tabular Reinforcement Learning is also an efficient framework to model human behaviour [9]. Therefore combining both may lead to very interesting results and Deep Reinforcement Learning could be a wonderful tool to better understand human cognition as it mixes decision making and neural representations, two fields that were already very prolific in last few years.

Before going in depth about the technical details let me first introduce few concepts about Reinforcement Learning and Deep Reinforcement Learning. After this I will go through different types of neural networks and their geometrical interpretation and then briefly present T-SNE, a method to visualize data from high dimensional spaces. I will then present the DeepMind paper and few ideas from this paper I wanted to deepen . Finally I will make a description of the experiment I carried and end with the results, limitations and what can be concluded from this work from a neuroscientific point of view.

2. Background

2.1. Reinforcement Learning

Reinforcement Learning is a sub field of unsupervised machine learning where the agent is given a goal without explanations about how to reach it and needs to learn to solve the given task.

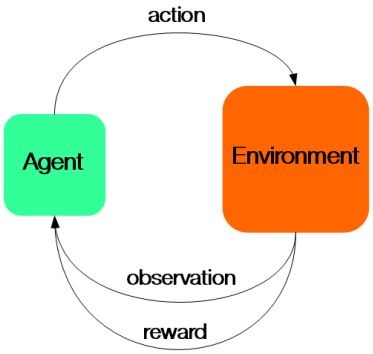


Figure 1: Agent-Environment system

More formally the agent learns in an environment which feeds it with observations to which the agent will react by choosing an action among a set of possible moves. Executing an action will modify the state of the environment yielding a reward and a new observation to which the agent can react etc... until the end of the trajectory 1. This environment is defined by a transition function $t(s, a, s')$ which outputs probabilities from going from state s to state s' by executing action a ($\int_s' t(s, a, s')ds' = 1$) and a reward function $r(s, a)$ which outputs the reward associated to executing action a in state s. This reward function can be a random variable in few scenarios (such as bandits). Trajectories are thus a sequence of $T = (s_i, a_i, r_i, s_i')_i$ of arbitrary length depending on the considered environment (with classical chess rules which avoid loops games always end and can have various length but theoretically the game can be infinite).

The goal of the agent given an initial state of the environment is to generate the trajectory returning the maximal reward. Given that a reward is given at any step in the trajectory it is important to define a score over the full trajectory. Most environment have finite length trajectories but to be mathematically compatible with infinite environment most people use a discounted return to score trajectories : $R(T) = \sum_i \gamma^i r_i$ with γ the discount parameter < 1.

2.1.1 Main concepts of Reinforcement Learning

To build agents capable of solving this kind of problem let me introduce few notations and formulas first. The *policy* is a function representing the agent's behaviour in the environment. In a stochastic setting this policy is the probability to choose action a in the given state s : $\pi(s, a)$. Learning the optimal policy is the final goal of RL and to achieve it efficiently most algorithms use value functions which makes it possible to better understand the dynamics behind the reward distribution over the Markov chain. These functions are always defined with respect to a given policy π. $V^\pi(s) = \mathbb{E}[R(T)|s_0 = s]$ is called the *value function* with T a trajectory starting from s sampled using π and $Q^\pi(s, a) = \mathbb{E}[R(T)|s_0 = s, a_0 = a]$ is the *action-value function*. We can define a partial order over value functions :

$$V^{\pi_1} \leq V^{\pi_2} \iff \forall s, V^{\pi_1}(s) \leq V^{\pi_2}(s)$$

With this order we can compare value functions and we have an interesting property :

$$\forall \pi_1, \forall \pi_2, \exists \pi_3, \forall s, V^{\pi_1}(s) \leq V^{\pi_3}(s) \cap V^{\pi_2}(s) \leq V^{\pi_3}(s)$$

This makes it possible to define the optimal policy for a given Markov Chain : $\pi^* = \arg\max_\pi V^\pi$. These equations are also true for action-value functions.

Moreover there are links between value functions and action-value functions :

$$V^\pi(s) = Q(s, \pi(s))$$

and for the discounted return :

$$Q^\pi(s, a) = r(s, a) + \gamma.\mathbb{E}[V^\pi(t(s, a, s'))]$$

Knowing t and r we can use these properties to find the optimal policy π^* with various algorithms such as Value iteration or Policy iteration [10].

Most of the time we don't directly have access to t and r and the agent needs to learn with a black box environment which only outputs observations and rewards during trajectories making exploration a crucial part of the learning process. Two approaches are then possible :

- Model-based Learning : In this settings the algorithm will try to learn unknown functions t and r through exploration and apply (or not) previous algorithms. This approach may not be very effective as the agent first needs to learn the dynamics of the environment before being able to improve its policy which can take a while and having an efficient policy is also important to reach relevant areas of the Markov Chain meaning at some point both will need to learn together often leading to chaotic results.

- Model-free Learning : Instead of learning a model of the environment we can directly learn the policy using samples taken from the environment like in Q-Learning [14] or SARSA [11] algorithms usually using an array to store value functions as well as the policy.

2.1.2 Deep Reinforcement Learning

In many environments the set of states and/or actions can be continuous/infinite making tabular approaches intractable. To tackle this issue the policy and value functions are approximated by various schemes such as neural networks.

The learning process will thus combine classical reinforcement learning (to decide what to learn) and deep learning (to decide how to learn it). This new family of algorithms could be seen as tabular reinforcement learning algorithm but with a gradient descent iterated over batches of trajectories on a neural network instead of storing values directly in an array.

The most basic algorithm of Deep Reinforcement Learning in probably REINFORCE [15] but many improvements either from a reinforcement learning or a deep learning point of view came to improve it like Advantage Actor Critic (A2C) [6] which uses the advantage function ($A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$) to reduce the variance of gradients using the baseline V or Proximal Policy Optimization (PPO) [12] which adds a policy regularisation on top of it in order to stabilize the learning of the policy.

2.1.3 PPO

For this project I will use PPO as main learning algorithm because its regularization term makes it very efficient in many environments. To go a little bit deeper in PPO here is the loss it uses to learn the policy network :

$$\begin{aligned} L1 &= ratio(s, a).A(s, a) \\ L2 &= clip(ratio(s, a), 1 - \epsilon, 1 + \epsilon).A(s, a) \\ L_{PPO}(\theta) &= \mathbb{E}[\min(L1, L2)] \end{aligned} \tag{1}$$

with

$$ratio = \frac{\pi_\theta(s, a)}{\pi_{\theta_k}(s, a)}$$

π_θ being the policy of the agent with parameters of the neural network θ being optimized, ϵ is a parameter of the algorithm (usually 0.1) and θ_k is the parameter of the network before fitting (it will fit several times on each batch) considered as constant in the gradient descent.

The idea behind this loss is that if the policy being fitted on moves too much on a given batch (overfit on the batch) the resulting behaviour will be unstable and will change too much between each batch leading to poor generalization. Here, if the policy changes too much and overfits on the given batch, the ratio will increase, be cut by the clip operator of $L2$ and the gradient will be cut by the $\min$ operator. Therefore this loss enforces the learning policy not be too much affected by a batch composed of outliers for example resulting in a more stable convergence of the policy.

Now that we have seen basic useful knowledge about Deep Reinforcement Learning let's talk a little bit about recurrent units and there geometrical interpretation.

2.2. Geometrical interpretation of Recurrent units

Recurrent units in neural networks makes it possible to add a time dimension to a neural computation through a latent space of *states* (not to be mistaken for RL states from the Markov Chain).

Elman RNN Elman RNN is a very simple recurrent unit :

$$h_{t+1} = \tanh(W_i.i_{t+1} + W_s.h_t + b) \tag{2}$$

with h_t the state at time t which is also the output of the layer, W_s the weight matrix associated to the state, i_t the input at time t, W_i the state matrix associated to the input and b a bias. Usually t starts at 1 and $s_0 = 0$.

Geometrically speaking at some timepoint t in the computation the state h_t is a point in a N-dimension space (N being the number of neurons in the RNN layer). This point will be modified by the W_s matrix which is the recurrent behaviour of the layer (counter increment, time increasing in a physics simulation moving objects, ...). Then the input produces a translation of this state $W_i.i_{t+1}$ with an additional constant translation b. In the end the $\tanh$ ensures the state stays in a specific domain.

This can be seen from a physics point of view : the system has an internal dynamics given by W_s and b and a controllable dynamics given by W_i and the input space. For example if the rank of W_i is lower than N, the system isn't completely controllable and few axes will only be influenced by the internal dynamics of the recurrent matrix W_s and the bias b. On top of it the activation function is a very strong force that keeps the dynamics in an arbitrary areas of the space.

To better understand how it works let's take an example : let's simplify a lot the system by saying that the states isn't too close to the borders of $[-1, 1]^N$ (thus we can remove the $\tanh$ as it is almost the identity function on this domain). In addition let's assume the recurrent dynamics is poor (time passing alone doesn't change the state - only the input does - which is often the case in many scenarios) we should have $W_s \approx I_N$ and $b \approx 0$ after training of the network. This means all the system does is translating the state each time a new input is given. Therefore the matrix W_i computes features of this new input and sends the state in the direction of these features and after few steps the state has accumulated all these translations and is returned. In this very simple example all the dynamics of the network can be understood by looking at components of the latent space and its geometrical transformations.

LSTM LSTMs are much more complex than Elman RNN but makes it possible to have a much richer dynamics.

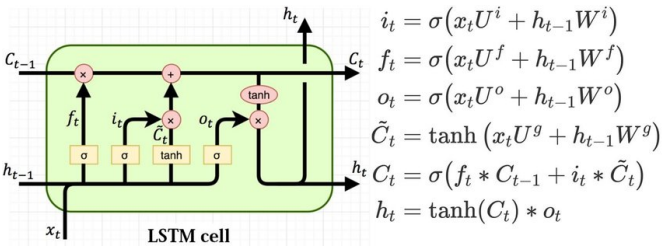


Figure 2: LSTM Scheme taken from [4]

In LSTMs there are 2 hidden states : c and h. The output of the layer is h only. Without going too much into the details, a LSTM is a Elman RNN with scaling operators which are the 3 gates (forget, input and output). The forget gate scales the c state at the beginning of the computation, the input gate scales the translation vector applied to c and the output gate scales c to generate the next h.

From a geometrical perspective this unit solves an issue of RNNs. If the new input is very informative and the new state of the network should be x to properly take into account this information (we will see later that it is often necessary to perform such an operation), the RNN knows how to translate in direction x from input i but does not necessarily know how to deal with the actual position of the state. That's why the dynamics is often not enough to reach the point x exactly. Here the scaling operator makes it possible to scale the actual state to 0 (closing the forget gate), then to translate in direction x (opening the input gate) and to return it (opening the output gate).

It is therefore a very powerful unit which has many interesting properties from a geometrical point of view.

2.3. t-SNE

When analysing data from a high dimensional space it is crucial to use relevant representation algorithms and PCA

often isn't enough. T-SNE [13] makes it possible to represent clusters from a high dimensional space in a plotable space (dimension 2 or 3).

The concept of the algorithm is first to compute a similarity matrix in the high dimensional space :

$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} \exp(-||x_k - x_l||^2/2\sigma^2}$$

This probability matrix represents the distance between each point in the high dimensional space.

Then to project it on a low dimensional space the algorithm randomly projects the points in a given space (usually dimension 2 or 3) and computes another similarity matrix :

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

The goal is then to enforce the two distribution to be as close as possible using a gradient descent on the KL divergence between both :

$$Loss((y_i)_i) = KL(p||q)$$

to find the best $(y_i)_i$ in the 2D/3D space keeping the same dynamics as the $(x_i)_i)$ in the high dimensional space. Using a gaussian in the high dimensional space encourages the generation of clusters in the low dimensional space. This distribution could be changed to use another prior.

3. Experiments

Now that we have reviewed most tools needed let me introduce the motivation of the experiments I worked on before going into the details of the work and experiments.

3.1. DRL & Neurosciences

A lot of work has already been done at the interface between Deep Reinforcement Learning and Neurosciences. DeepMind published a paper few years ago making a review of many different ideas that were studied recently mixing Deep Reinforcement Learning and Neurosciences [3] I want to discuss a little bit.

General similarities The main idea developed and shown under many different lights in this paper is that if Deep Learning and Reinforcement Learning are both useful to model cognition, Deep Reinforcement Learning is more than just the concatenation of both. In fact Deep Learning brings a lot to reinforcement learning generalization purposes because Tabular Reinforcement Learning for example cannot generalize to new states close to the one previously seen which is a field in which Deep Learning is already very effective. Additionally Reinforcement Learning which makes value functions and the policy learn together helps the Deep Learning aspect with for example actor critic models which often keep a shared part of the network for the actor part and for the critic part to encourage one to use the features developed by the other which often increases the performances of the network.

On the other hand Deep Reinforcement Learning is a field very close to neurosciences and psychology as almost all algorithms and improvements developed in Deep Reinforcement Learning have an interpretation in those fields. For example in [9] they have shown that in humans, the value function isn't learnt in absolute but in relative value which they modelled by the REFERENCE framework. On the other hand in Deep Reinforcement Learning the REINFORCE model had variance issues in its loss which has been solved by A2C [6] and PPO [12] by using the value function as baseline to normalize the loss. This mimics the findings of the cognitive science paper. These two discoveries were made in parallel (none of them knew each other) meaning Deep Reinforcement Learning researcher made discoveries in Neurosciences before neuroscientists without even knowing anything about neurosciences. In that case, could it be possible that neuroscientists are actually making the future of Deep Reinforcement Learning as well ?

In addition to the way Deep RL algorithms are designed, networks often show the same activation patterns than humans in terms of memory and representations.

Memory and representations The question of memory is central in Deep Reinforcement Learning and Neuroscience and I would like to emphasize few interesting points developed in the paper that I wanted to deepen:

- First, most common recurrent units in Deep RL are LSTMs and those gates could have similar properties to neurons in the brain. I wanted to investigate the difference between networks having gates and Elman RNNs to see if indeed there is a huge difference in representations.
- Second, they suggest the representations in the latent space of the policy seem to cluster actions and the latent space for the value function clusters observations by values. This is also something that I wanted verify and develop on the specific case of spatial tasks.

3.2. Environment

To study the spatial representations I chose to work on discrete mazes. Those mazes 6a are composed of 2D cells (squares) and the agent can move from a cell to a connected adjacent one. It starts on a random cell in the maze and needs to reach the exit (that is defined at the beginning of the training and never changes). The agent only sees its local context (the connection of its actual cell to the 4 others

$\in \{0,1\}^4$, 1 for possibility to reach the cell and 0 if there is a wall) and gets a +1 reward if it reaches the exit and a $-\epsilon$ reward at each step ($\epsilon = 0.001$ in my setup). From this setup I expect the agent to learn to solve the specific given maze and to understand the connectivity between cells to reach the exit.

The algorithm used to generate mazes is Prim's algorithm which generates a maze from a breadth first search of the grid. It is important to notice that this algorithm generates labyrinths and not mazes (there are no cycles). It could be interesting to introduce cycles to see how the agent reacts to this but I didn't do it in my research.

This task is very hard for a RL agent as it requires a lot of exploration (sparse reward) meaning that without any improvement in the pipeline the convergence can be quite long. For example it is possible to use curriculum learning [8] [2] to faster the training. However these methods change the way the environment is shown to the agent by modifying the initial position distribution starting from a simpler distribution and slowly making it reach the desired one which might change the way spatial representations are developed. Once more this could be very interesting to study but that wasn't the point of my project. I thus took a maze size of 17x17 (see Fig6a) because that was the maximum maze size I have been able to train agents on.

The point of this environment is that it is large enough for multiple cells to have the same observation meaning the agent needs to build a map of the maze to know where it is and where it should go. At the beginning it could even need few steps before figuring out where it is in the maze. This agent will therefore need to have a recurrent memory in order to solve this task efficiently.

3.3. Pipeline

To learn such a dynamics we need to use recurrent neural networks. Usually in DRL people use a recurrent embedding at the end of the network (the layer right before the output layer) but it forces the recurrent latent space to have almost linearly decodable representations which may be a quite strong prior. Instead I have put this layer in the middle of the network 3 to have a first embedding by 2 FC layers and then a decoding by an other set of 2 FC layer before getting policy probabilities and values. This structure makes it possible to have less linear and possibly more complex and more effective spatial representations in the network.

As recurrent units several choices are possible. Most of the time people doing DRL use LSTMs as they make it possible to carry information on the long run along trajectories. That's why I have used LSTMs and Elman RNNs to compare the representations developed by both recurrent cells.

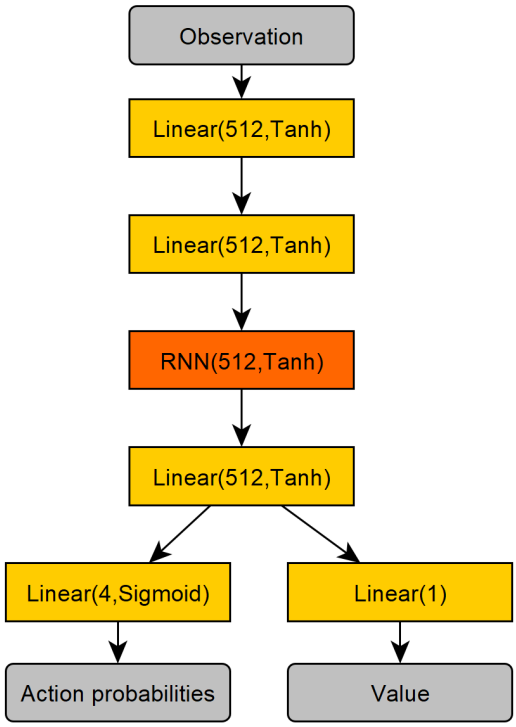


Figure 3: Scheme of the network used (the RNN layer can be switched to a LSTM layer)

3.4. Results

The training were made using RLLIB [1] which provides a set of state of the art DRL algorithms with a parallel implementation which makes it a reliable library to train agents.

Running a training with a RTX 3080 GPU only on the hub and i9 processor (10 cores) with 10 remote workers takes approximately 10 hours and leads to an optimal behavior of the agent.

3.5. Analysis

Once we will have trained agents we will be able to take a look at what is happening in the neural network.

To visualize neural representations I put the agent somewhere random in the maze, made it find the exit using its knowledge acquired during training and recorded the activity of the state of the recurrent layer. For each step of each trajectory I thus got a vector of high dimension (usually 512) which represents the neural activation. It is then possible to plot those trajectories in a low dimension space using T-SNE [7] (PCA isn't powerful enough to visualize such complex spaces - probably because the relevant information may not be linearly encoded due to the shape of the network). With this process I have been able to register

1000 trajectories to visualize.

Trajectories in the latent spaces are neural representations registered at each cell of this trajectory. During this trajectory, from the agent point of view, many interesting features can also be recorded to analyse neural representations and find correlations with what it contains such as time (progress in the trajectory - can be seen as the log of the value), position in the maze, action taken from the current cell and the full length of the trajectory (maybe long trajectories aren't stored similarly to short ones).

The maze considered is represented in 6a and a color code is used to represent each part of the maze in the next T-SNE plots 6b.

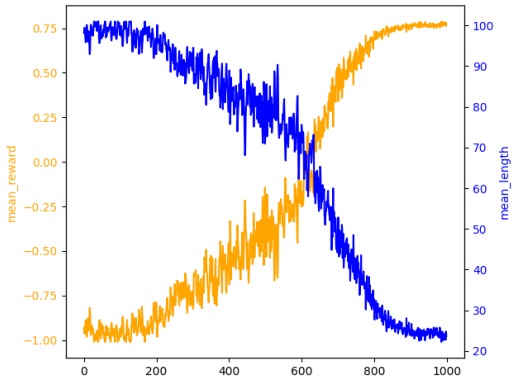


Figure 4: Mean reward and trajectory length during training (these are training values, not testing ones)

Elman RNNs Training with an Elman RNN brings a final score of 0.8 (mean reward per trajectory) for a mean length of 20 which is almost optimal 4. Elman RNNs have a state of size 512 which makes things simple. The 2D T-SNE algorithm on these trajectories returns an interesting representation that we will analyse under the light of the 4 quantities discussed before.

There are three interesting things to notice in those plots.

- First the neural representation seems to have built a mental map of the maze as clusters of the same color can be seen in 7b. Moreover each cluster seem to output the same action 7c and same value 7a which is coherent with the DeepMind paper [3]. However we see that end of trajectory clusters (red clusters) seems to be much more consistent on the outer edge of the plot. On the other hand blue clusters are more at the center of the figure and aren't very clear (cf Fig 7b we don't clearly see if they are clusters linked to position).

  This could be due to two things :

  – first the cells far from the exit (on the border of the maze) are less visited meaning they are under-represented in the trajectory set. In addition every trajectory starts somewhere random but ends at the exit (and goes through the same end cells) meaning the red part of the maze is over-represented in the data provided to T-SNE. The algorithm should be robust to this (more than PCA for example which outputs the time axis as the major variance axis) as long as enough data is provided for each cell to draw clear clusters.

  – Another possibility is that these cells are much less visited than cells close to the exit so the agent might not know them very well and their representation could be unclear and mixed.

  To know which of the two hypothesis is right we could run the algorithm with much more datapoints. This way if the representation is fine, having more datapoints would make the algorithm cluster much better these cells and we should get a figure with very clear clusters for each cell of the maze. Nonetheless T-SNE needs to compute similarity matrices which cost a $O(n^2)$ complexity for n datapoints given. With 1000 trajectories the algorithm is fed around 20 000 datapoints which makes the conversion already pretty long so I couldn't add more points to verify this hypothesis.

- Second interesting thing to notice is the fact that cells that are close to each other in the maze seem to not be close in the representation space.

- Third very interesting thing to understand is the first few steps in the environment. Looking at blue points in Fig 7a we see that the bluest points are either spread around the figure or at the center of the figure. The first few steps in the environments are characterized by uncertainty. The agent needs to understand where it is in the maze only getting its immediate neighbourhood as information. Thus to know where it is and where to go to find the exit it needs to move several times and aggregate those observation to find the unique place in the maze which can combine all these local observations. Therefore, when looking at representations, those points seem to correspond to this searching phase of the very few steps where the agent needs to find its location. After those steps the neural trajectory goes into a known cluster and ends like all the others. What is interesting here is that this searching phase isn't clustered : the agent knows it doesn't know where it is and we see in 7b that in those states the agent can be anywhere in the maze and take any action 7c.

LSTMs Training with an LSTM brings a final score of 0.82 (mean reward per trajectory) for a mean length of 18 which is a little bit better than RNNs 5. LSTMs have 2

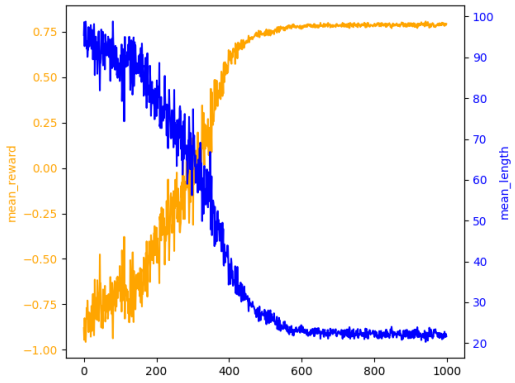


Figure 5: Mean reward and trajectory length during training (these are training values, not testing ones)

states of size 512 which work together (information might be redundant between the two) so I have concatenated those 2 states in 1 state of size 1024 (and using PCA or T-SNE should remove the redundancy).The T-SNE algorithm on these trajectories returns an interesting representation that we will analyse under the light of the 4 quantities discussed before.

In these plots we remark similar things with the RNN plots except for few interesting details.

- The first point is similar : the neural network made a mental map of the environment, cells seem to be clusters in the latent space but still the center is a a little bit unclear. We find the same insights given in the DeepMind Paper [3] with value and actions quite well clustered (around cells). For the same reasons than for RNNs it's hard to put more trajectories to verify if this unclear central representation is due to the agent not knowing exceptionally well the borders of the maze or if it's noise due to the low number of cells in trajectories sampled in those areas.
- On the other hand representations seem to be a little bit closer in space that for the RNN. It's still far from perfect but its a lot clearer.
- Lastly their is a huge difference when it comes to how the network handle starting representations. With the RNN, first states are not clustered which shows the network uncertainty about where it is. Then, the activations align with a pattern representing a position in the maze and the agent can smoothly solve the maze. Here on the contrary there are successive clusters of uncertainty which as a decision tree slowly orient the activations to the right cluster. We clearly see it in Fig 8a with blue circles which like a tree structure orient the activations. This cluster is very sharp in the T-SNE projection (meaning states are very close in the latent space), corresponds to multiple positions 8b and actions highly depends on the input as we can see in Fig 8c because even if states are very similar multiple actions can be taken which clearly discriminates those clusters from the other ones.
- A last point which can be made about LSTM neural representations is the fact that clusters, despite being clear for most of them aren't as sharp as the RNN clusters. It seems to exist a certain variability in LSTMs for a given position in the maze. This variability could be explained by an additional variable such as the trajectory length (see Fig 8d) as it seems to vary inside clusters but it's not very clear.

3.6. Discussion

Even if it is hard to speak about how the maze is globally stored in the neural representations we can assume an embedding of cells is done and is efficient at least in the cells in which the agent often goes. However close cells in the maze do not have close representations in the embedding of the neural network (at least for RNN and it seems to be a little bit more accurate in the LSTM but not very clear).

On the other hand the way first steps in the maze are embedded is very clear between both models. The RNN seems to encode those first steps far from each other probably taking much more memory of the latent space to encode more information (maybe store all observations seen explicitly until it has enough to decide where it is ?) where the LSTM does it in a more compact way by building a decision tree structure. This ability to decide much more efficiently is probably linked to the ability of LSTMs to forget specific data meaning it can easily switch from a representation to another which is harder to do with a RNN as explained earlier.

3.7. Limitations

The first limitation of my work is that I didn't verify if the results still stand with a different maze (training and analysis scripts take a while to run) which is vital to know if these observations are consistent with different mazes.

Then LSTMs have a larger state (2 states of 512) meaning it cannot be directly compared with RNNs which only have 1 state of size 512. We could put twice more units on the Elman RNN network to keep the same state dimension but it would mean more weights in the RNN network which may also wrong the conclusions. That's why I stayed with the same number of units despite this difference.

Lastly observations made in this work are based on a visualization technique which represent data from a space of dimension 512 (1024 for LSTMs) to a space of dimension 2 meaning observations could be wronged by the representation , to be sure about results, statistical computations

should be performed in the full sâce to ensure about how well clusters are significant, about whether the sparsity of activations for the first few steps of RNNs is consistent and not bad luck with the way data are initially projected on the low dimensional space, ...

4. Conclusion

In the end this experiment on spatial representation yields 2 main results :

- On one hand I have been able to observe few insights given by the DeepMind paper [3] about the clustering of actions and values in the latent space (both at the same time because I used a network which shares representations which is quite interesting as those clustering merge very well).
- On the other hand representations developped by RNNs and LSTMs are similar on the big picture (clusters of positions and actions) but they handle the start of the task differently due to the forget gates of LSTMs and it could be a very relevant task to get human fMRI data to validate the fact that humans' recurrent activations are closer to LSTMs than Elman RNNs (but I don't have access to such data so I leave it to competent people in this field).

Lastly here are a few ideas to go further :

- It would be interesting to test the generalisation power of different representations by modifying the position of the exit in the maze once the agent is trained and to see how long it takes for it to learn how to solve this maze with a new exit position. This could be a particularly relevant task to study because Deep Reinforcement Learning is supposed to be better at generalization than Tabular or Linear Reinforcement Learning as exposed in the DeepMind paper [3].
- To investigate even more what happens at the beginning of the task when the agent needs to find its position in the maze we could use bigger mazes. However I couldn't do more than 17x17 for technical reasons. We could go beyond this limit using curriculum learning and study its impact on representations development as well.
- Lastly the DeepMind paper suggested that Model-based RL creates representations which are closer to humans than Model-free RL. This could be an interesting project to see if representations change between both models.

References

[1] Ray rllib. `https://github.com/ray-project/ray`, 2017.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. 2009.

[3] Matthew Botvinick, Jane Wang, Will Dabney, Kevin Miller, and Zeb Kurth-Nelson. Deep reinforcement learning and its neuroscientific implications. 07 2020.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[5] Tim Kietzmann, Courtney Spoerer, Kenneth Sörensen, Radoslaw Cichy, Olaf Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116:201905544, 10 2019.

[6] Volodymyr Mnih, Adrià Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 02 2016.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[8] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey, 2020.

[9] Stefano Palminteri and Maël Lebreton. Context-dependent outcome encoding in human reinforcement learning. *Current Opinion in Behavioral Sciences*, 41:144–151, 2021. Value based decision-making.

[10] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem. 1996.

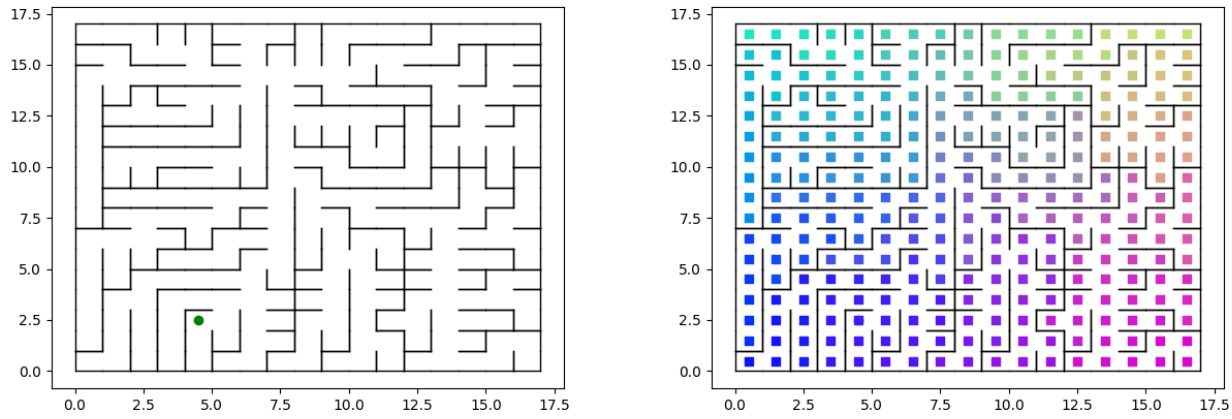
[11] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[13] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[14] Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992.

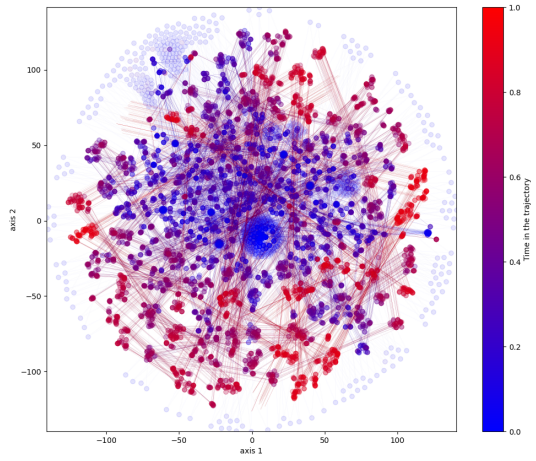
[15] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.



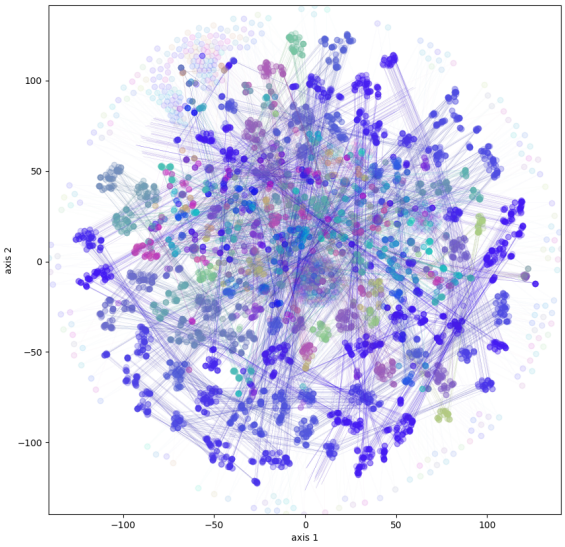
(a) Maze with exit (green dot).

(b) Maze with colors to visualize position in next plots

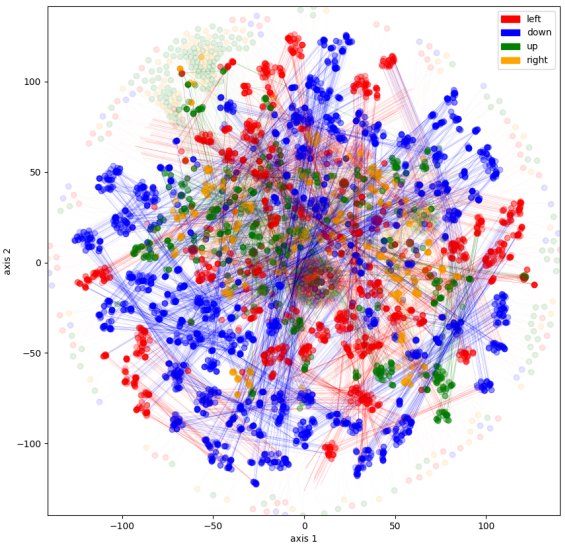
Figure 6: Maze considered for this experiment. The agent spawns somewhere random and needs to find the exit getting only local information about its surroundings.



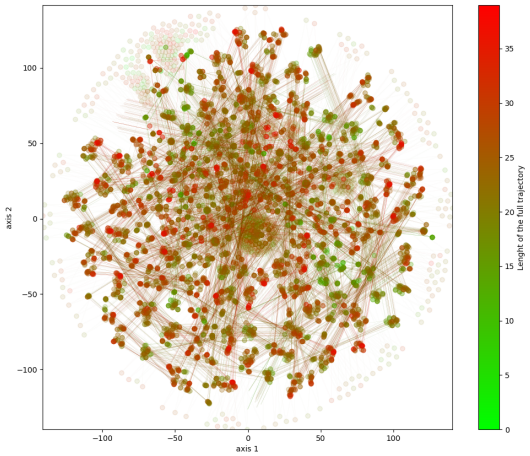
(a) Time



(b) Position

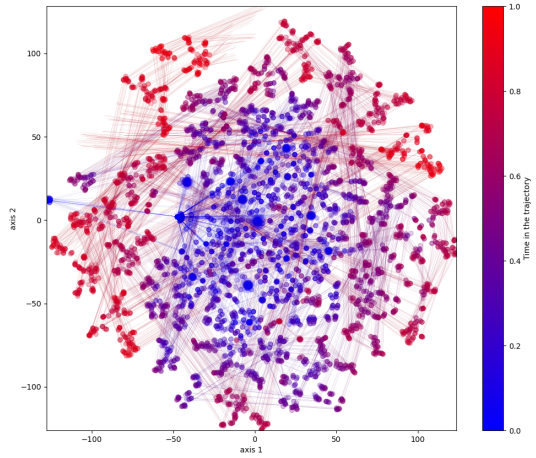


(c) Action

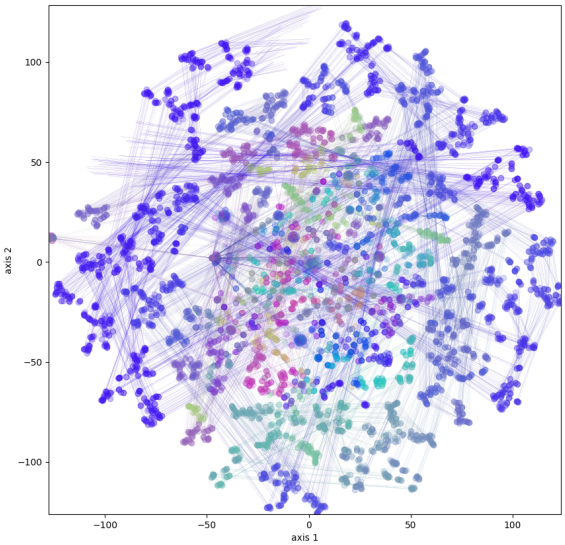


(d) Length

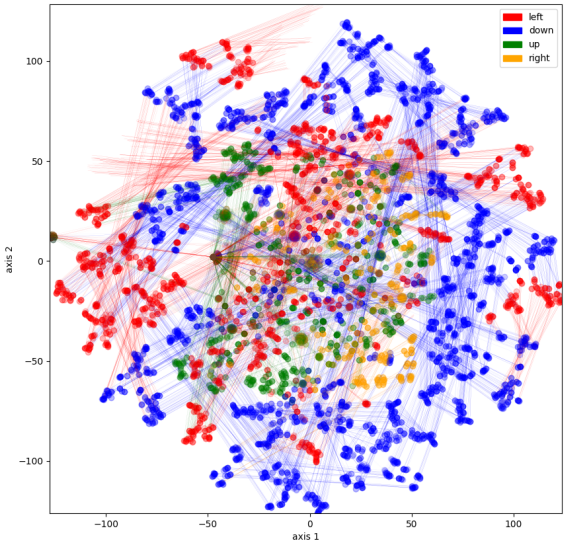
Figure 7: Analysis of the model using a RNN with T-SNE



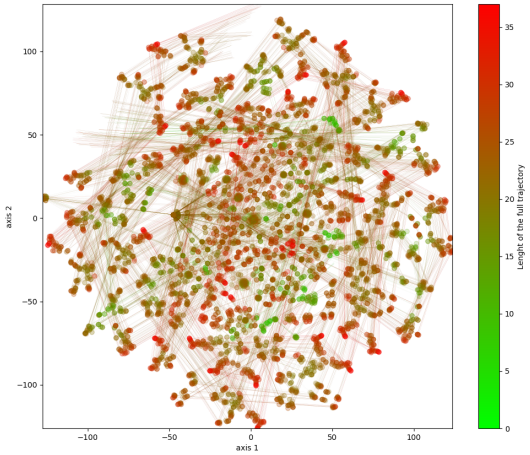
(a) Time



(b) Position



(c) Action



(d) Length

Figure 8: Analysis of the model using a LSTM with T-SNE