IMPROVE NEURAL NETWORK LEARNING PROCESS IN REINFORCEMENT LEARNING FIELD BY ACTIVATION FUNCTION PROJECTION

Chupin Clément
Student in Master degree
University of Clermont-Ferrand
clement.chupin.si@gmail.com

March 1, 2023

ABSTRACT

In reinforcement learning (RL), understanding the observation space is a key point for all algorithms, feature extraction can improve the convergence and final performance of the RL agent, especially in robotics. Moreover, real-time applications have specific requirements that can be satisfied by feature extraction, such as a good sample efficiency or a good generalization.

In this paper, we study the feature extraction in the reinforcement learning field by projecting our input into different activation functions (Triangular, Gaussian, Sinusoidal), we try to determinate the advantages and disadvantages of the activation function projection, we present multiple feature extraction experiments. We aim to exploit most variants of input pre-processing, applied to the most of classical policies (on/off policy, continuous and discrete action space), to the most of tasks (Mujoco, OpenAI classical gym control).

Finally, we will see that these techniques are promising, but that there is no general method to apply and take advantage of them in the case of reinforcement learning.

1 State of the Art

In recent years, machine learning has received a lot of attention due to its ability to extract patterns and in formations from a complex environment. A critical aspect of machine learning is the selection and pre-processing of the input data, which plays a crucial role in the performance and accuracy of the neural network. Input transformations are one of the key techniques to improve the performance of machine learning algorithms. We apply a series of mathematical operations (commonly neural layers) to the input data to extract features and patterns that can improve the accuracy of the final model. There are different types of input transformations in various fields of study, such as computer vision, natural language processing or signal processing. Input transformations can be used to improve dimensional understanding, reduce the size of hidden layers, and improve the data efficiency of our neural network. In this project, and in the field of reinforcement learning, we would like to increase the understanding of the RL agent, to have better sampling efficiency and better performance on the task. In other words, we will try to modify our RL agent to learn faster and better to complete a task.

1.1 Reinforcement learning in the robotic field

Traditional robotic programming involves a lot of manual programming to control the robot's actions, which can be time-consuming and often requires expertise in robotics and programming.

Reinforcement learning is a type of machine learning that is particularly useful in robotics, as it allows an agent (such as a robot) to learn to perform tasks in complex and dynamic environments (such as walk). Reinforcement learning teach to an agent how to make decisions in an environment in order to maximize a reward, that correspond to a specific task. In the context of robotics, reinforcement learning is a powerful tool, it can be used to train robots to perform complex tasks and make decisions, allowing robots to learn and adapt to a dynamic and unpredictable environment, to perform complex tasks with a increased efficiency.

Finally, RL can be used to train robots to perform tasks that are difficult or impossible to program manually. For example, in an agricultural environment, a robot may need to adapt to changes in the task process or environment, and RL can help the robot learn to adapt in real time.

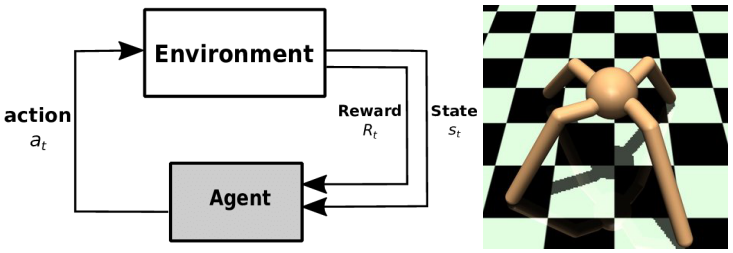


Figure 1: Representation of the learning process, for an ant who learn to walk

All environments used in this project are listed here A.

The input data in reinforcement learning includes the agent's state and reward information, which can be transformed using various techniques to improve the agent's performance. In this project, we will explore various input transformations, and their gains on the agent's performance (succeed to accomplish a task, to learn faster...).

1.2 Feature extraction

In machine learning, feature extraction is a process of transforming raw data into features that can be processed while preserving the information in the original input. It's a key point of every machine learning process, a good understanding of our input is an advantage to predict an accurate output. It can lead to a better understanding of the input by the neural network, and finally, the RL agent can learn faster and better to solve a task. This process might be compute by specific layers.

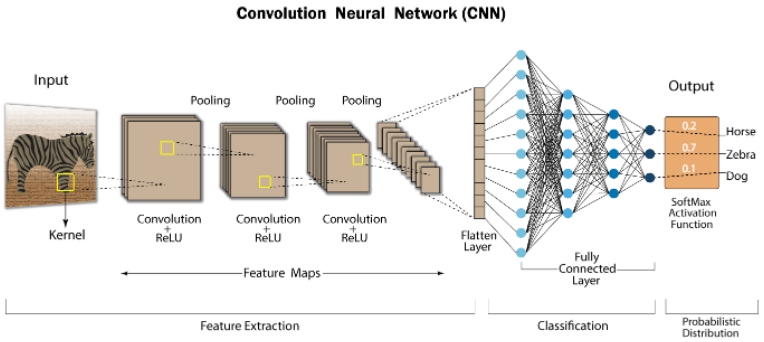


Figure 2: Example of feature extraction with an MLP for image classification in supervised learning

In this case, we try to recognize different animals based on RGB picture of them. Recognize pattern, specific characteristics of animals, and the final MLP classify animals, based on the feature extraction. The neural training is possible by the supervised task, train on the dataset, the convolutional layers will be able to recognize specific pattern, and the classify MLP will learn to recognize animals, based on the feature recognition.

1.3 Convolutional layer apply to image processing

In many cases, in the image processing, we have to compress the data in a latent space to be able to accomplish our task (such as classify, segment, tracking objects...). A direct process of the image can be impossible, it's too big and too complex to the MLP to succeed to understand the raw data.

In a convolutional layer, the neural network applies a set of filters to an input image to extract specific features, such as edges, corners, and textures. These filters are learned during training through the learning process of the neural network (backpropagation, gradient descent).

As the filters are applied, they create a set of feature maps that highlight different aspects of the input image. The resulting output from the convolutional layer is a set of high-level features that represent the input image. These features are then passed to the subsequent layers of the neural network for the final task. Neural networks can automatically learn relevant representations, without the need for a manual tuning.

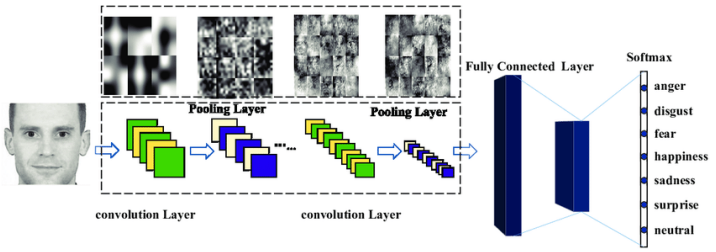


Figure 3: The general structure of convolutional neural network. Convolutional neural network mainly includes convolution layer, pooling layer, and full connection layer. The convolution layer is responsible for feature extraction, the pooling layer is used for feature selection, and the full connection layer is used for classification. [1]

1.4 Autoencoders feature extraction

In an autoencoder, the feature extraction is performed by a bottleneck layer that compresses the input data into a lower-dimensional representation (latent space). The bottleneck layer is generally composed of one or more convolutional or fully connected layers.

During training, the autoencoder learns to reconstruct the original input data from its compressed representation, by minimizing a reconstruction loss between the input and the output. The autoencoder learns to encode the most relevant information about the input data into the latent space.

Once the autoencoder is trained, the latent space can be used as an easier input for predict an output. By using the autoencoder to extract features, it is possible to learn a representation of the input that captures its most important features.

Autoencoders are particularly useful to deal with a high-dimensional data, such as images or text, where manually designing feature extractors can be challenging and time-consuming.

Feature extraction can play an important role in reinforcement learning by reducing the dimensionality of the observation space and providing the agent with a more concise and informative representation of the environment. But there is a problem, the autoencoder can only perform compression when the data is compressible, that means the data must present a link between the different inputs. This is why autoencoders are particularly used in image processing.

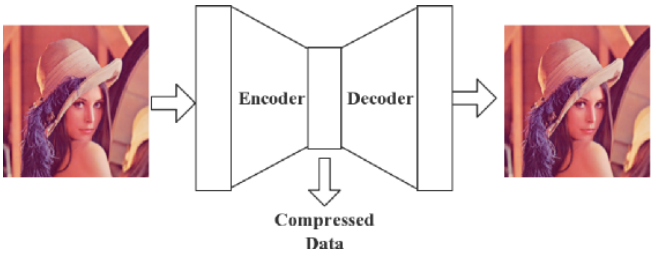


Figure 4: Autoencoder representation

1.5 Feature extraction in the reinforcement learning field

Convolutionnal layers is often used to image processing with pattern, autoencoder can be extend to every data who can be compressed (which present pattern), both are useful for a neural network to work on a latent space with a better representation.

In many reinforcement learning problems, the state space can be large, making it difficult for the agent to learn an effective policy. Feature extraction can help address this issue by reducing the dimension of the state space, while retaining the most relevant information of the task .

For example, in a robotic navigation task, the state space might include raw sensor measurements, such as camera images and LIDAR readings. These sensor measurements can be high-dimensional and contain a lot of redundant or irrelevant information. Feature extraction can be used to pre-process the sensor measurements and extract a smaller set of features that are more informative for the navigation task, such as the distance to obstacles, the direction of the goal, and the robot's velocity.

But for a lot of reinforcement learning task, specially robotic task, the input data can't be compressed in a latent space to increase the dimensional understanding of our agent. The input is about the state of the robot, which can be completely independent, which is impossible to predict based on a smaller representation. This is a part of our project's problematic, a generic feature extraction can't be applied to a RL agent in the field of robotic to increase its performance on a task.

1.6 Pre-processing in the machine learning field

We will focus on specific transformations applied to an input that can be used by the neural network to have a better understanding and accomplish its task. We will see different ways for a neural network to pre-process an input, to, based on raw data, extract a feature. The goal is to increase the understanding of the network, by activation function projection, to explain, link and extract features in the different inputs. The output of the preprocessing is not necessarily smaller and more representative than the input, it can augment the data without providing new understanding, this will be the role of the neural network.

1.6.1 Activation functions

An activation function helps to take decisions in a neural network by applying a non-linear transformation to the output of each neuron. This non-linearity is important because it allows the neural network to model complex relationships between the inputs and outputs. Without an activation function, a neural network would simply be a linear regression model, which is limited in its ability to represent complex patterns in the data.

The activation function also helps to introduce non-zero gradients that allow the neural network to learn and improve through backpropagation during training. In other words, the activation function will modify its parameters to reach the ideal transformation of its input, according to the learning process.

1.6.2 ReLU activation function

The ReLU (Rectified Linear Unit) activation function is a popular choice in neural networks because it can help to perform prediction on the input data.

When an input is passed through a ReLU activation function, any negative values are set to zero, while positive values are retained. This effectively eliminates any negative information from the input and preserves only the positive features. This can be seen as a form of feature extraction, where the most relevant and informative features of the input are isolated and passed on to the next layer of the network.

Additionally, ReLU has a simple and efficient computation, making it well-suited for use in deep neural networks, where large numbers of neurons and layers are common. Its piece-wise linear nature also makes it easy to optimize using backpropagation, which is an algorithm used to train neural networks.

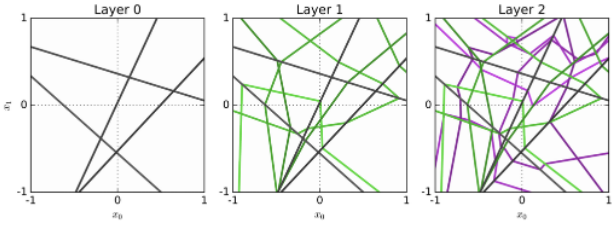
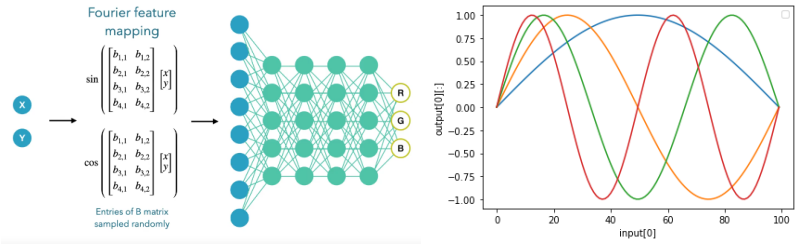


Figure 5: ReLU input decomposition, [2]

This is the natural process of predict an output based on input, in all feature extraction, we impose specific operation to our input, who can be view as determinate neural layers. In our study, our activation function are usually fixed, contrary to the ReLU function with custom parameters, funded in the learning process of a neural network. We had the idea to have a first layer with a fixed ReLU, but without specific objective, it's inefficient, the ReLU have to be learned to be efficient to represent data.

1.6.3 Fourier Features

Fourier Features from (Tancik et al, [3]) :



Representation of the Fourier Features extraction, [3]

The Fourier Features is a transformation of the input to have a better representation for the neural network, the input is convert in a biggest expression space, by multi-frequency representation of each input. We will project the input on different determinate Sinus function to have a better dimensional understanding, and the classical MLP after that will learn in a classical SGD to obtain a solution. The projection help to have a better dimensional understanding because a couple of input (a*x1 + b*y2 ...) will be projected with every possibility, the low frequency and the high frequency will work together to give to the MLP a better dimensional representation.

The simplified formula is :

$$FF_i(X) = \omega((X_0 * K_{0,i} + X_1 * K_{1,i} + ... + X_n * K_{n,i}) * 2\pi) \tag{1}$$

Refer to G to more informations.

We have tested this method on image reconstruction :

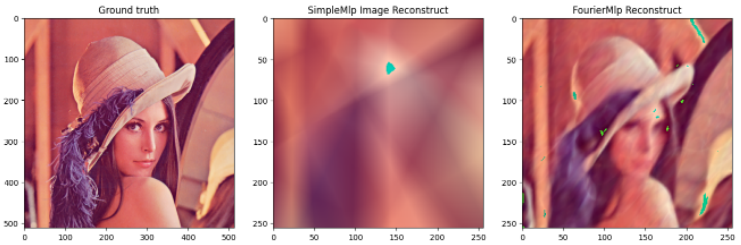


Figure 6: Result of the different methods of image reconstruction F

We can see that the Fourier Features helps the neural network to have a better dimensional understanding, and based on the position, have a better prediction of the color of a pixel.

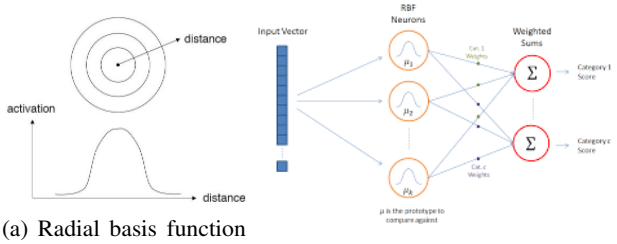
1.6.4 Radial Basis Function Kernel

Radial basis function (RBF) is a type of kernel function used in machine learning to transform input data into a new feature space. RBFs are commonly used in supervised and unsupervised learning for classification, regression, and clustering tasks.

The RBF kernel function is defined as a distance-based function, which measures the similarity between two input data points. The RBF kernel function takes an input vector and computes the distance between the input vector and a set of reference vectors. The output of the kernel function is a scalar value that describes the similarity between the input vector and each reference vector.

In this way, RBF can be used to describe an input by measuring its similarity to a set of reference vectors in a higher-dimensional space. This can be useful in various machine learning applications, such as clustering or anomaly detection, where it is important to group similar inputs or identify outliers based on their distance from a set of reference vectors.

In a supervised case, we can learn to neurons to detect a specific range, based on the data :



(a) Radial basis function representation

(b) RBF used in a neural network

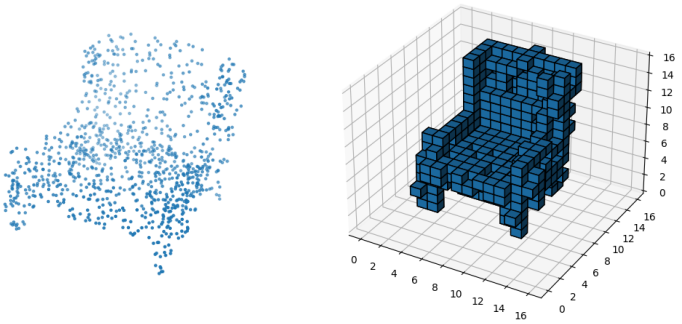
Figure 7: Explanation of Radial Basis Function in a neural network

Based on our knowledge, radial basis function is used in supervised learning, unsupervised learning, but for the reinforcement learning field, RBFs are used to learn the weights of the neural network, to process an output, but not to process a feature extraction, the direct use of RBFs is reserved to supervised and unsupervised learning.

1.6.5 Voxelization

Especially for dealing with 3D point clouds, we can reduce our input by voxelization. This technique is to be used on known and comparable data, a compression through the unsuitable dimensions can deteriorate the understanding of the neural network.

The voxelization on the point clouds reduce the number of points involved and attempt to getting a faster and more efficient network. Voxelization implies that the points are placed in voxels. For each voxel, the points centroid is kept as a new point and all centroids form the new point cloud.



(a) Raw 3D point cloud (b) Voxelized 3D point cloud

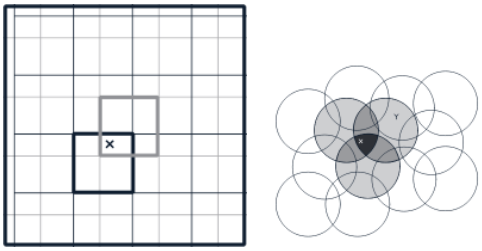
Figure 8: Representation of a 3D point cloud

1.6.6 Coarse encoding and Tile coding

Similar to the voxelization, but with a general application, Coarse encoding divide the input range into a fixed number of bins and representing each bin with a single binary value. This approach can help the neural network to better understand the input data by converting continuous values into discrete, binary values that can be more easily processed. Coarse encoding can also help to reduce the dimensional of the input space, which can improve the efficiency of the neural network.

Tile coding is a more sophisticated approach that involves dividing the input range into overlapping tiles, each of which is represented with a binary value. By using overlapping tiles, Tile coding can provide more fine-grained representations of the input data than coarse encoding, which can lead to better performance in some cases.

Tile coding can also help the neural network to better understand the input data by providing more detailed information about the input space. This can be particularly useful in applications where the input data is continuous, such as in robotics, where the different position and orientation of the robot can be represented with continuous values. By using Tile coding, the neural network can better understand the robot's state, which can help it to take decisions.



(a) Tile coding (b) Coarse encoding

Figure 9: Representation of feature extraction by dimensional coding

Unfortunately, there is a dimensional problem in these methods, it's the dimensional exploding if our task have a lot of input, each input will add a new dimension in our decomposition. We can see the problem as an N-D grid where every title is a neurons receptor, if we have 10 dimension for 5 subdivision, we have 5^{10} 10 billions of neurons. In our case of robotics task with a continuous observation space, we will use independent sensor neurons, the rest of the neural network will have the job to combine different input and find a solution to accomplish the task.

According to Sutton and Barto :

"On tasks with very high dimensionality, say hundreds of dimensions, tile coding and RBF networks become impractical. If we take either method at face value, its computational complexity increases exponentially with the number of dimensions. There are a number of tricks that can reduce this growth (such as hashing), but even these become impractical after a few tens of dimensions." [4]

This have to focus our attention on a problem for the reinforcement learning field, we have to choose technics which can handle a adaptive number of input. The algorithm need to have a maximum complexity of $O(n)$, in contrary of $O(n^2)$ or $O(2^n)$ which tend to be impossible to compute when the numbre of inputs increase (n).

1.6.7 Kanerva coding

Kanerva coding is a technique that can help a neural network to better understand and work with high-dimensional input data. This approach is based on the idea of representing input data as a sparse binary vector, where only a small number of the elements are active (have a value of 1) and the rest are inactive (have a value of 0).

In Kanerva coding, the input space is divided into a large number of small hypercubes, each of which is represented with a binary vector. When an input falls into a hypercube, the corresponding vector is activated, while all other vectors are set to 0. By using this approach, Kanerva coding can provide a more fine-grained representation of the input space than other encoding techniques.

Kanerva coding can be particularly useful in applications where the input data is high-dimensional, such as in natural language processing or image recognition. By using Kanerva coding, the neural network can more easily process and understand high-dimensional input data, which can lead to better performance and accuracy. By representing the input data as a sparse binary vector, the neural network can more easily process and store the data, which can help to reduce computational and memory requirements.

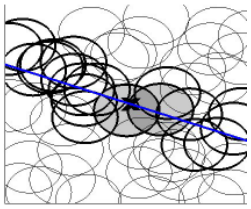


Figure 10: Representation of Kanerva coding, the active dimension is represented by the blue line

"A more useful way to think about the problem is to focus on the complexity of the target function as separate and distinct from the size and dimensionality of the state space. The size of the state space may give an upper bound on complexity, but short of that high bound, complexity and dimension can be unrelated. For example, one might have a 1000-dimensional task where only one of the dimensions happens to matter. Given a certain level of complexity, we then seek to be able to accurately approximate any target function of that complexity or less. As the target level of complexity increases, we would like to get by with a proportionate increase in computational resources." [4]

Kanerva have a very advantageous theoretical solution, but it is based on a delicate assumption, that some dimensions of our problem are not very important. This is a problem for reinforcement learning, first of all, we don't know which input is more important than another, and determinate that is already a part of the reinforcement learning. Finally, we potentially need absolutely all the inputs to be able to adapt the agent to the task, even if this technique were effective in simulation, it would be problematic in the real world to forget one of the sensors.

1.6.8 Bspline base function

An inspiration for our project is the Bspline function, which is in robotics optimization field, a way to describe the time to simplify the problem, contrary to find a solution from the begin to the end of the task, we solve it by interval. We define a multitude of function which focus on a specific areas of our time, and the combination of each of them lead to determinate a global solution.

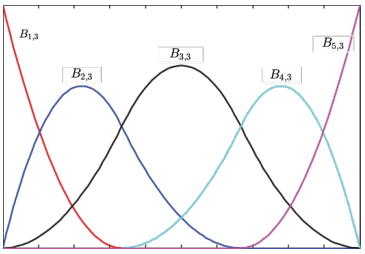


Figure 11: Bspline functions

1.7 Conclusion of current feature extraction method

In reinforcement learning, feature extraction is the process of identifying and selecting relevant informations from the raw input data that can be used to make decisions and take actions. While feature extraction is a common technique in machine learning, it can be challenging in the context of reinforcement learning, especially in the field of robotics.

We can describe different types of data who permit to have a feature extraction :

- For recurrent data when we can identify pattern like images, we can use convolutional layer, or autoencoder to extract the main informations of the data.

- For dependant data, when we want to cross different dimensions, we can use Fourier Feature, tile coding or coarse encoding, but these technics have a several limitation, like for a particles filter, the computation is exponential with the number of dimensions to compute ($O(order^{dimensions})$).

To take in count this problem, we can use Fourier Light Features, or Kanerva coding, the cross between different dimension is let to the MLP processing, we aren't limited by the number of dimensions (in our project, the number of input our the RL agent).

All these pre-processing bring a lot of solutions, but specially for supervised and with low dimensional problems, or for problems with bigger dimensional, different coding bring solutions to determine features when the information is recurrent and compressible. To a RL robotic task, we have more inputs, which are independents, and the output isn't a reading of our feature space but a prediction of actions based on the properties of our robots.

One of the main difficulties with using feature extraction in reinforcement learning for robotics is that the data cannot be compressed. This means that all the raw sensor data must be processed and used to train the reinforcement learning algorithm, which can be computationally expensive and time-consuming.

In general, for a reinforcement learning task, if the inputs are correctly constructed, the RL agent succeeds in reaching its task, but sometimes the MLP fails to learn.

2 Study

This project tries to determine the advantages and disadvantages of activation function projection pre-processing of the input, compared to classical MLP (no input pre-processing), with a evaluation of the difference in reward and convergence (how much the agent succeed the task, and how fast it learn), for a robotic task in the reinforcement learning field.

Regarding real time, with this study we hope to bring RL algorithms into new real time tasks, especially in robotics, which require good sampling efficiency and generalization with limited hardware. Due to the cost of sample acquisition, safety and robustness of classical automatic engineering, the use of neural networks in direct control is not yet common, augment the understanding of the agent can be a good point to facilitate the introduction of RLs in real life. One assumption we made is that adding a layer of linear transformations at the beginning of the agent, if it helps to have a better understanding, does not have a significant computational power cost adding of the RL agent computation.

2.1 Problematic

Describe in our state of art, a lot of technics exist to help a neural network to understand its input, but for the reinforcement field, applied to robotic, when every input are important, it's harder to find a way to have a better understanding of the input by the agent.

Some recent search tend to solve these problems, by the using of Fourier Features to increase the dimensional understanding [5]. A potential critic to this work is than sinus function is hard to compute, compared to ReLU or triangular, and that, for reinforcement learning, we don't need to increase the dimensional understanding but we need to increase the areas understanding. Fourier Features [3], and more generally sinus function[6], aim to reconstruct data with a high levels of details, like images, song, 3D shape. So, the technics is focus on the dimensional understanding in despite of the areas understanding, this is why the Fourier Features help the agent in the RL, the dimensional understanding help to have a better areas understanding.

In this project, we would like to find an algorithms to pre-process the input of a RL agent, to have a better understanding of the observation space, to finally succeed to solve the task faster and better.

2.2 Policies and Environments

For the environments, we use OpenAI Gym [7], for the policies, we use OpenAI Stable Baselines3 [8]. This two libraries are developed by OpenAI and are, together, perfectly compatible.

We call "Environment" for a specific task to accomplish by a reinforcement learning agent, like driving a car, control a robot, play a video game... (all environments used are listed here A).

To our policies, the purpose is to test the most popular implementations, with the most basics and commons neural networks after the feature extraction. So we will keep the default neural network architecture provide by OpenAI stable-baselines3 [8]. A training policy concern two parts of the reinforcement learning process, how the agent will interact with the environment (make a decision based on the observation of the environment), but also how it will be trained (based on the rewards, how the task is solved, change the parameters to obtain a better decision process). The architecture and hyper-parameters might change between different policies (refer to the HP of each policy in E).

In this study, we aim to evaluate our algorithms on a representative panel of tasks generally treated in reinforcement learning, specially robotics tasks. Moreover, we use different policies of learning to evaluate our input pre-processing. Indeed, many of studies focus on resolution of video games, which is relevant in the research in the field of autonomous artificial intelligence, but in our case, we focus on the resolution of physical problems.

2.3 Activation function projection

In the context of machine learning, data augmentation is the process of creating new training examples from existing data by applying transformations or perturbations to the input data. One common way to perform data augmentation is to project the data onto different activation functions. This can help to solve the problem of feature extraction in reinforcement learning by allowing the neural network to learn more complex representations of the input data.

When the data is projected onto different activation functions, the neural network is exposed to a wider variety of input data, which can help it to learn more complex features. For example, a triangular activation function may capture information about local trends in the data that a standard activation function such as ReLU may miss.

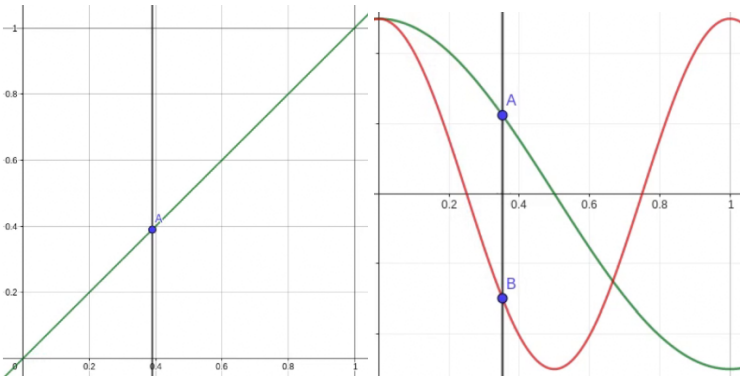
By training the neural network on a variety of different activation functions, it becomes more robust to different types of input data, which can improve its performance in a variety of different tasks. This is especially true in reinforcement learning, where the environment can be dynamic and unpredictable, and the input data can vary widely depending on the situation.

Overall, projecting the input data onto different activation functions can help to solve the problem of feature extraction in reinforcement learning by allowing the neural network to learn more complex representations of the input data. This technique can be especially useful in robotics applications, where the input data can be complex to understand.

First of all, before describe the different feature extraction, we will explain the principe of activation projection function

$$FeatureExtraction(\begin{bmatrix} in_{(0)} \\ in_{(1)} \\ ... \\ in_{(n)} \end{bmatrix}) => \begin{bmatrix} f(in_0, 0) & f(in_0, 1) & ... & f(in_0, order) \\ f(in_1, 0) & f(in_1, 1) & ... & f(in_1, order) \\ ... & ... & ... & ... \\ f(in_n, 0) & f(in_n, 1) & ... & f(in_n, order) \end{bmatrix} \tag{2}$$

We project our input into functions to obtain the output :



(a) Linear Projection (b) Multi-Frequency Projection

Figure 12: Representation of the activation function input projection

We have the linear input decomposition (or no feature extraction), we send a value, we receive the exact same value, it don't change the data.

Also we have a representation of the Fourier Light Features, a decomposition who increase the dimensional understanding by projecting input into different frequency of a sinus. The low frequency will focus on general areas of the input, the high frequency will focus on the details, by the combination of each, we have a way to describe with high levels of details an input (Tancik et al., [3]).

Finaly, for our study, we will focus on activation function which help to have a better areas understanding of our input, we will project on the same activation function, but each of them focus a specific areas of the input :

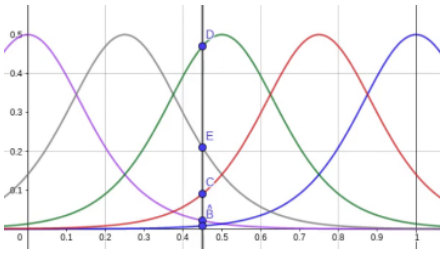


Figure 13: Gaussian mutli-areas projection

For example, with Gaussian decomposition, each Gaussian will focus on a specific area of our input, and act as a field receiver. Close to the radial basis function but with a deterministic variance and mean of each Gaussian.

All the layer projections used in this project are listed here D.

This idea was inspired by biology, to copy the mechanism of the brain understanding a biological input, the brain doesn't receive a raw information ("your arm is touched at position 0.57 meters 0.59 meters, with a pressure of 0.42 Newton"), but a multitude of neurons ("I receive a multitude of neurons, they said (0.01 volts, 0.1 v, 0.4 v, 0.1 v), I think I am hit on my hand with a pressure of 0.5 Newton").

To copy this mechanism, we try to convert a linear input into different neurons, the problem with these methods is that the MLP loses the position information of each neuron, it cannot take advantage of it, the MLP forgets that one neuron is close to another, it receives two independent neurons. A convolutional layer could be a good solution, but it is superfluous, compressing the number is useless, and a better mixing can be obtained by increasing the variance of the Gaussian.

We will focus on the triangular activation functions for two reasons :

- The computational gains of using the triangular function compared to the gaussian function.

- The biggest difference between the two is the gradient, the triangular does not give a "good" gradient, one that learns the weights correctly, but in our study we determined that the decomposition should be fixed, not learned, so we don't take in count the gradient.

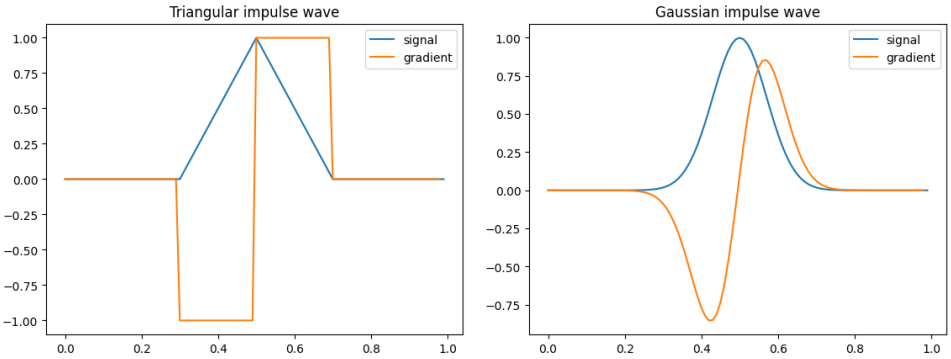


Figure 14: Signals and their gradient

2.4 Exploration

The exploration is the biggest part of this project, we have tried a lot of technics, in different variations of them, we have to run our experiment on every environment, for every policy, to see if the technics can be use full in almost every case, and evaluate the results.

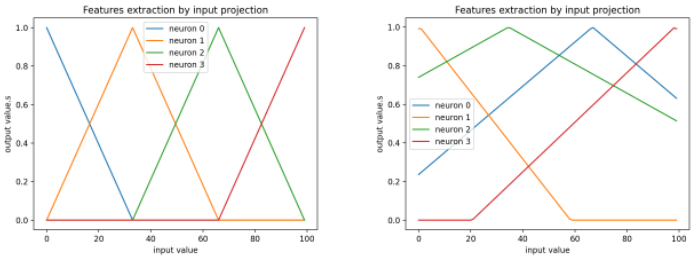
For a clarity of the report, and by computational limits, we do not show the result of the exploration, because they give a result similar or worse than the final experient.

2.4.1 Triangular activation function

First was triangular activation function projection, close to the ReLU, we have the triangular decompostion, fixed and learned. The fixed will describe the environment by areas, the learned will find itself (in theorie), the specific areas to focus.

Using a triangular activation function for data augmentation is an interesting approach, but its effectiveness may depend on the specific task and the characteristics of the input data.

A triangular activation function is a type of piece-wise linear function that is defined over a triangular region. It can be used to introduce non-linearities into the neural network's output and can help to capture complex patterns and relationships in the data.



(a) Fixed Triangular (b) Learned Triangular

Figure 15: Triangular activation function projection representation

We can see this decomposition as a cutting of our input, we have tested this method on image reconstruction :

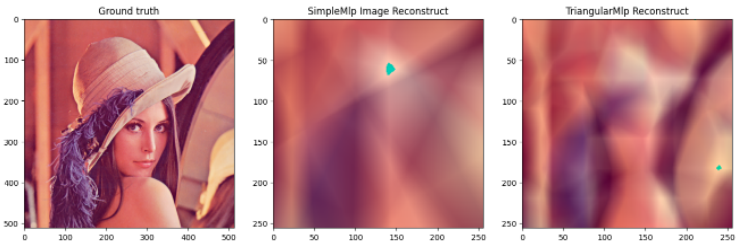
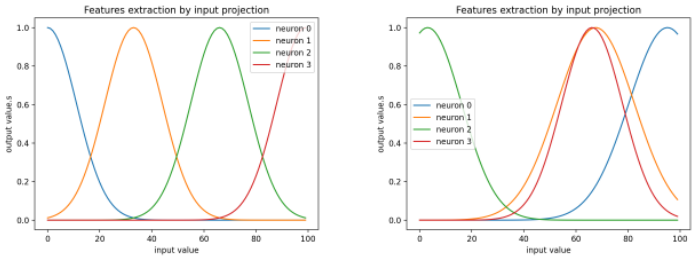


Figure 16: Result of the different methods of image reconstruction F

This was the first experiment, the fixed triangular activation was promising, it help in some case to have a better understanding of our input. For the learned triangular, it don't show interesting result, just worst than a classical or fixed triangular.

2.4.2 Gaussian activation function

We have also tested, similar to the radial basis function, the Gaussian activation function projection. Harder to compute but with a smoother shape than the triangular, it might bring a better solution :



(a) Fixed Gaussian (b) Learned Gaussian

Figure 17: Gaussian activation function projection representation

Results was similar to the triangular function, the learned was not promising, and the Gaussian result very similar to the triangular, promising but not consistent. This can be explained by the visual shape, the triangular and the Gaussian are very close, the biggest change is on the gradient, and it don't show improvement than the learned triangular activation.

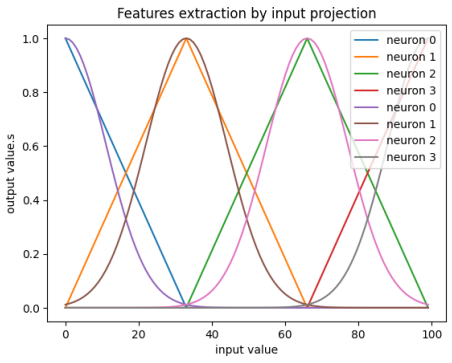
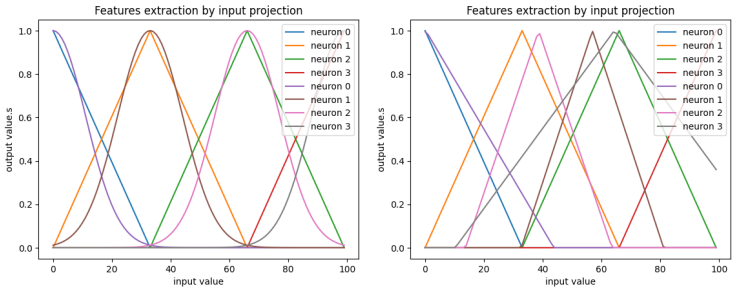


Figure 18: Comparison between Gaussian and Triangular activation function

2.4.3 Mixture of activation function

We tried to combine different activation function projection to see if we can take advantage of each of them and get a better result :

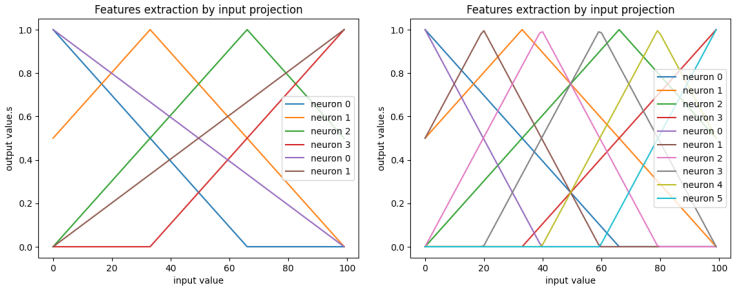


(a) Fixed Gaussian and Triangular (b) Fixed and learned Triangular

Figure 19: Cross-over between activation function projection

After experiments on combination between fixed and learned, triangular and Gaussian, we don't succeed to find a general way to extract the input to help the reinforcement learning agent.

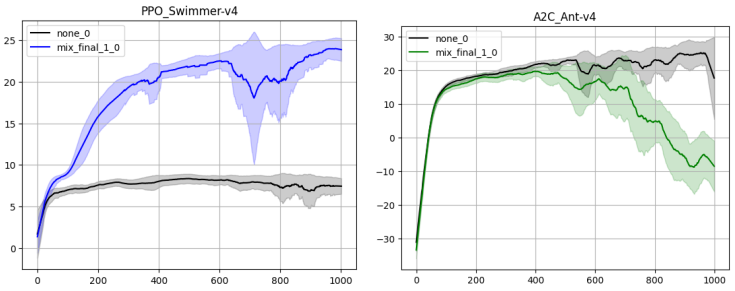
Finally, after remove Gaussian and learned activation function, we focus on mix triangular activation function. The idea was to give a maximum of information to the neural network to improve its understanding of the task.



(a) Triangular with order of 2 and 4 (b) Triangular with order of 4 and 6

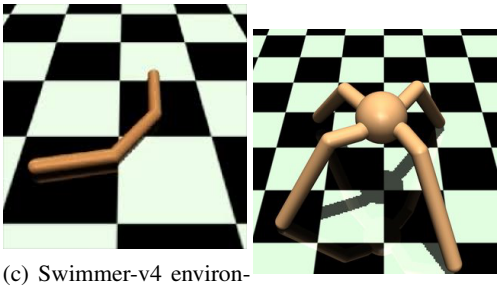
Figure 20: Mixing between triangular activation function projection

The results was promising, but we can't see a general way to how to use this technics and have a agent with a better understanding of the task and how to reach it. It is promising because, in despite that the pre-processing have often a worst result on the learning process of our RL agent, sometime, the pre-processing help to learn faster and better, we hope to find a generic pre-processing for every policy and every task.



(a) Good learning result

(b) Bad learning result



(c) Swimmer-v4 environment

(d) Ant-v4 environment

Figure 21: Example of how the RL agent take advantage or disadvantage of the pre-processing

We can see that for the good result, the pre-processing help the RL agent to learn faster and stronger to accomplish the task, we can conclude that the RL agent have a better understanding its input which permite to reach it. For the bad result, the RL agent can't take advantage of our pre-processing. It's frustrating, for the same processing, for a similar task, the RL agent have a completely different result, it's the empirical part of the machine learning field.

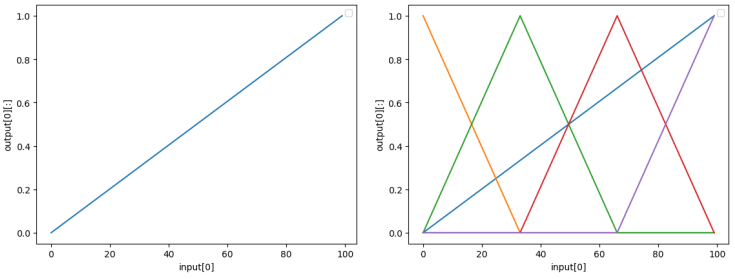
2.5 Evaluation of different Triangular activation function projection

At the end of our exploration, we don't succeed to find an ideal method, so, to present our work, we have focus on Triangular activation function projection, and Fourier Light Features with triangular function to re create the sinus function (TLF : Triangular Light Features in our experiment).

2.5.1 Triangular activation function projection

After a lot of promising experiment, we will implement a general input decomposition, based on triangular decomposition to be able to give the result of the most usable version of the triangular activation function projection.

First, we will construct our decomposition based on a linear function (the mix is called mix_final_1).

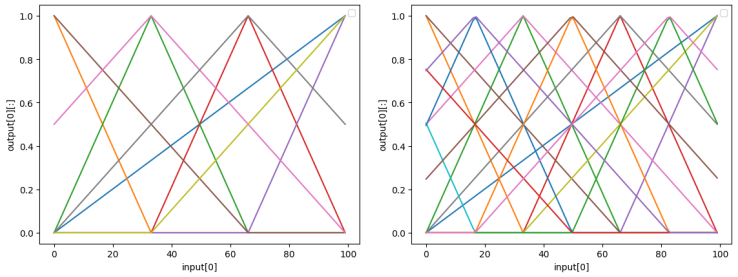


(a) Linear

(b) Linear + Triangular

Figure 22: Construction of a feature extraction by linear and triangular activation function

Finally, to take in account different variance and phase, we create mix_final_2and mix_final_3.



(a) Mix with variance (b) Mix with variance and phase

Figure 23: Construction of a feature extraction by linear and triangular activation function

After building this pre-processing, we will test it on different policies and tasks, with 5 experiments each time to have a representative result. This will allow us to see the advantages and disadvantages of the pre-processing technique. Indeed, due to the random nature of reinforcement learning, we need to test the technique, both in several possible configurations, but also several times on the same configuration. The choice of 5 epochs is debatable, with the limited computational power, and our result which was not innovative in the reinforcement learning field, 5 epochs is a good compromise.

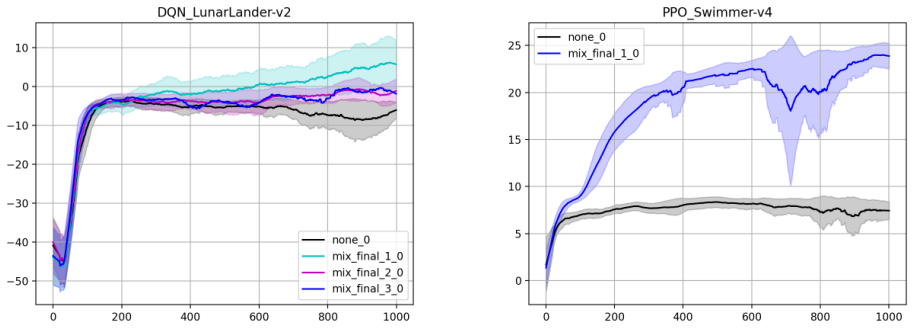


Figure 24: Best result for a specific environment and policy, mean on 5 epochs

We can see that the LunarLander environment with a DQN policy, and the Swimmer environment with a PPO policy, have a good response to our pre-processing. This is a good start to understand why and how our pre-processing helps the neural network to accomplish its task, but in our case, these two configurations have a good response to almost any pre-processing that increases the dimensional understanding. In order to find a general pre-processing, we evaluate it on each configuration (environment/policy).

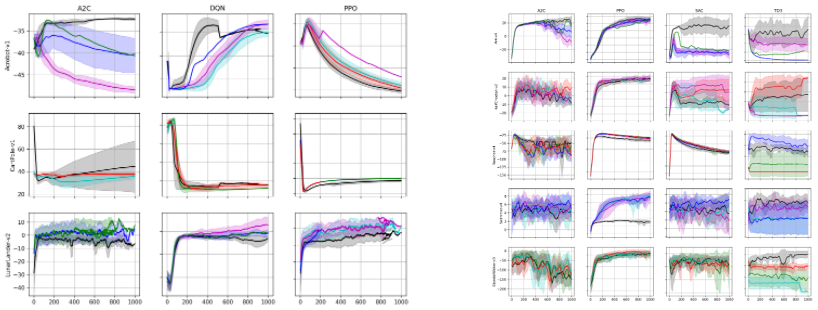


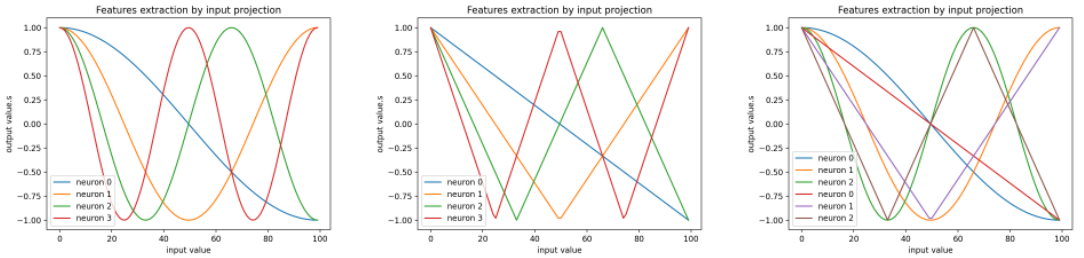
Figure 25: General result of the mix triangular, B

Unfortunately, the result of our pre-processing can increase the learning speed and strength of a reinforcement learning agent in specific task and policy, but its effectiveness can't be universally applicable. With no general application, we can't present the research community with a novel approach to decomposing an input.

2.5.2 Triangular Light Features

The triangular activation function present another interest than the areas decomposition : a fast execution. A limit of the application of Fourier Feature may be the computation requirement, be able to replace a sinus by a triangular function can be useful in this field.

To be able to see the difference between a sinus function and a triangular function, we try to reproduce the work of David Brellman [5], we succeed to have similar results, but with a different neural network computation, there are not exactly the same. The repeatability is respected because our result are coherent with the original paper, if we execute the exactly same software, we certainly have the same result.



(a) Sinus function (b) Sinus with Triangular function (c) Sinus and Triangular function

Figure 26: Visualization of differences between Sinus and Triangular function

The use of triangular functions to approximate a sinusoidal signal is not directly related to reinforcement learning, but rather to signal processing and control systems. In reinforcement learning, the goal is to learn an optimal policy for an agent interacting with an environment, typically using trial-and-error experience.

Approximating a sinusoidal function using a triangular function is possible, but the approximation may not be very accurate. We test it on image reconstruction, we have very similar between sinus and triangular function (losses and visual reconstruction, we have tested it on a small order of decomposition to highlight the advantages and disadvantages of each pre-processing).

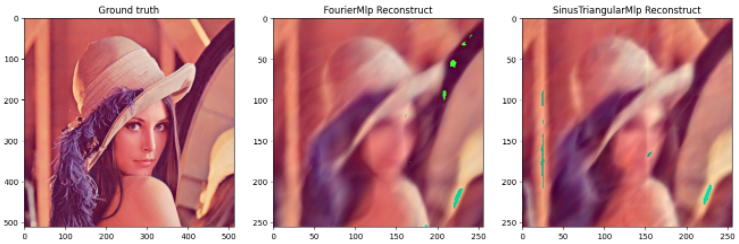


Figure 27: Result of the different methods of image reconstruction F

To highlight the interest of the triangular activation function, we reproduce a sinus by a triangular function. If these method is similar to a sinus, it can help to have a better computation and apply the method with a limited hardware in real-time.

First, we have the result of the Fourier Light Features with different order of decomposition :

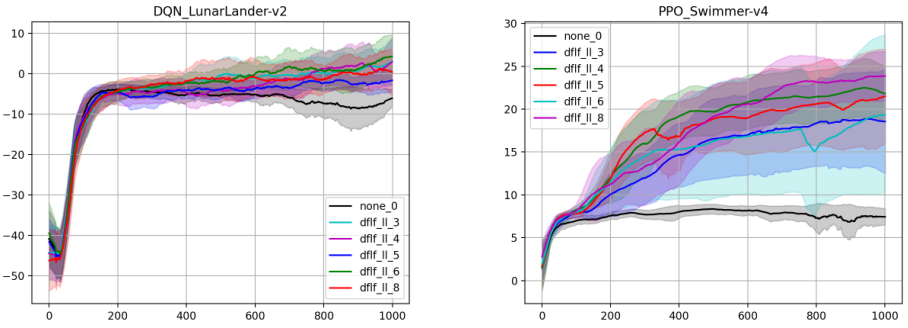


Figure 28: Best result for a specific environment and policy, mean on 5 epochs

Like said before, these two configuration have a good response to pre-processing which increase the dimensional and/or areas understanding.

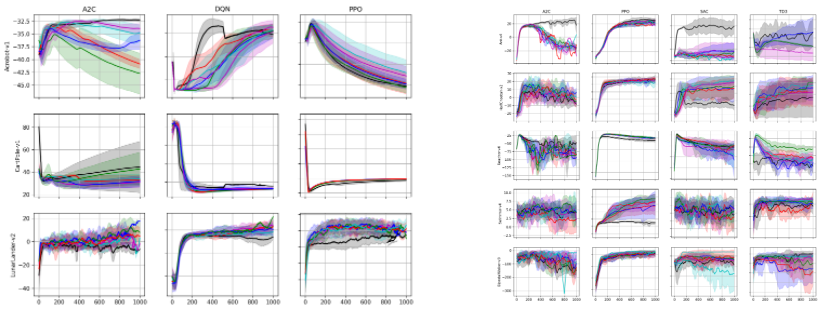


Figure 29: General result of the Fourier Light Features, B

Fourier Fight features show very promising results for reinforcement learning, although this technique was created for signal processing ([3]), recent work has shown great interest in using this technique to increase the dimensional understanding of the agent, in the field of reinforcement learning ([5]).

Finally, we have the result of the Triangular Light Features with different order of decomposition :

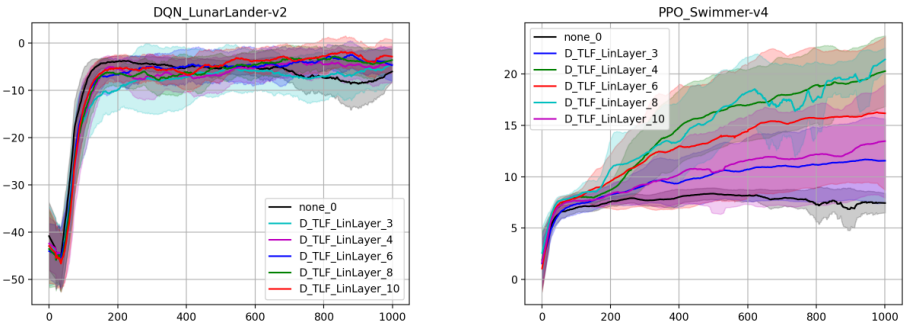


Figure 30: Best result for a specific environment and policy, mean on 5 epochs

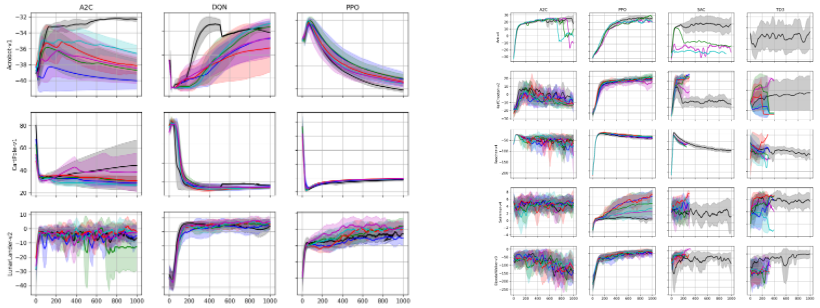


Figure 31: General result of the Triangular Light Features, B

We can see that we have similar result, but not exactly the same convergence. For now, it's too early to be able to replace the real sinus by a triangular function, we already have to find and test technics based on Fourier Light Features in the reinforcement learning field before extend the triangular function to facilitate the computation.

For the supervised and un-supervied field, the research turns to the learned Fourier features, in other words, a decomposition into different learned frequencies, so the gradient of our activation function is very important, and despite the fact that the sine and the triangular are close, their gradient is completely different, we cannot currently replace a sine with a triangular.

In reinforcement learning, the RL agent responds differently to the triangular function and the sine function, like for the supervised and un-supervised learning, one can't be replaced by the other.

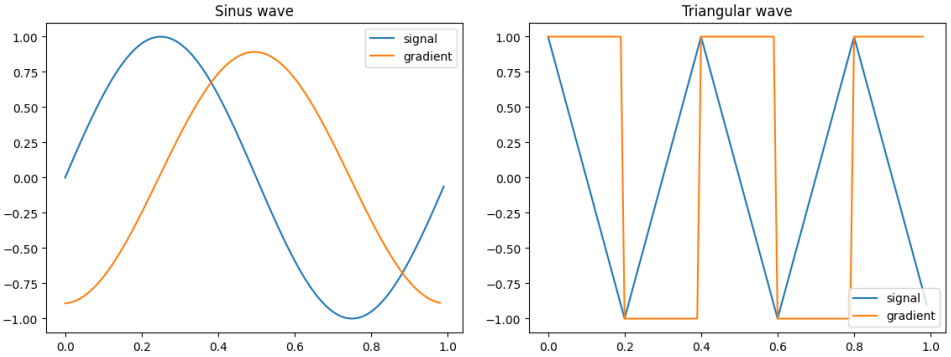


Figure 32: Signals and their gradient, the sinus signal gradient is a cosinus, the triangular signal gradient is a square signal

2.6 Conclusion of experiments

The triangular activation function in reinforcement learning field for robotics, has shown promising results in providing better understanding of the input, but unfortunately, there is currently no universally applicable and conclusive solution to the problem. Continued research and experiments is needed to develop and refine the method for optimal performance in robotics (specially [5] for the Fourier Features in the RL).

Regarding non-conclusive results, we can hypothesize that, in some cases, a triangular activation function may introduce unnecessary complexity into the model and cause over-fitting, resulting in poor performance on new data, the RL agent fails to understand the input and reach its task. In other cases, it can be a useful tool to improve model performance by allowing it to capture more complex patterns in the data.

In general, the choice of activation function should be based on experiments and evaluation of model performance on training and validation data (simulation and "real" environment for RL). While a triangular activation function may be a useful technique in some cases, it is not a universal solution and should be used judiciously.

3 Conclusion

3.1 Problematic

Our research did not lead to a universal method for projecting an activation function that would improve an agent's performance, but we identify a potential replacement for the sine function and highlighted the possibility of improving the agent's understanding of a zone in specific cases via the triangular activation function projection.

This project highlighted the specificities for a reinforcement learning agent to learn from a modified input. Indeed, the transformation of the input, as we have empirically approached in this project is delicate, ideas of transformations can be conclusive, especially when we know the nature of the data (image processing, point clouds, signals...). But when we don't know the nature of the data, the result of the transformation is empirical, it can work, it can't work, there is no general rule behind it.

In the case where we are looking for a generic transformation, affecting a variable input, it is much more difficult to find a transformation that allows the agent to learn better to perform its task. This study also allowed us to highlight that each reinforcement learning task reacts differently to the learning policy.

This project highlights the importance of further research and experiments in this area to improve the performance of reinforcement learning agents in various applications. Overall, the results of this study contribute to the growing body of knowledge about the complexities of reinforcement learning and the strategies that can be employed to improve agent performance.

3.2 Result

Our results did not live up to our expectations: we did not succeed to find a method that would allow an agent to have a better learning process through a better understanding of its environment. However, we did manage to highlight the possibility of having a better understanding of a zone, in specific cases, via the triangular activation function projection. This study has demonstrated that a tailored approach may be necessary for each reinforcement learning task due to the unique ways in which they react to algorithms.

While the results did not meet the initial expectations, the study did reveal the potential for a better understanding of specific zones through the use of triangular activation function. Additionally, the study highlights the possibility of replacing the sine function with a triangular function to enable the use of Fourier Features in cases where computing power is limited.

3.3 Open problems and future research

Moving forward, further research is needed to develop a universal method for input projection and determine the most effective use of linear function. The findings of this study provide the need for continued research to improve the performance of reinforcement learning agents.

Looking forward, future research in this area could focus on developing a more comprehensive understanding of the impact of input modifications on reinforcement learning agents. This could involve exploring different types of input transformations and identifying the most effective approaches for improving agent performance. Additionally, in the field of reinforcement learning research, this project highlights the need to study different tasks and environments, so that a new approach can be proposed to other researchers.

This project taught us the right scientific approach, the importance of defining a specific problem and using analytical methods instead of relying solely on empirical approaches. Rather than trying to find a quick fix for every problem, we should focus on a specific issue, a problem that many people are facing, and try to solve it. Finally, we think that our mistake was to launch ourselves into a subject, where the problematic was not well defined, with a subject much more complex than we thought, and with a multitude of possible solutions, which lost me more than helping to advance in a clear direction.

4 Special thanks

I would like to thank Mr Romuald Aufrère for the supervision my Master degree, and for the validation of this study project, as well as Mrs Céline Teulière for the supervision and her help in this project.

Computations have been performed on the supercomputer facilities of the Mésocentre Clermont Auvergne. We are grateful of their help.

References

[1] Yingying Wang, Yibin Li, Yong Song, and Xuewen Rong. Facial expression recognition based on random forest and convolutional neural network. *Information*, 10:375, 11 2019.

[2] Ryan Jeong, Will Barton, and Maricela Ramirez. Final report: Deep neural networks. 2020.

[3] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *CoRR*, abs/2006.10739, 2020.

[4] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. 2015.

[5] David Brellmann, Goran Frehse, and David Filliat. Fourier features in reinforcement learning with neural networks, 2022.

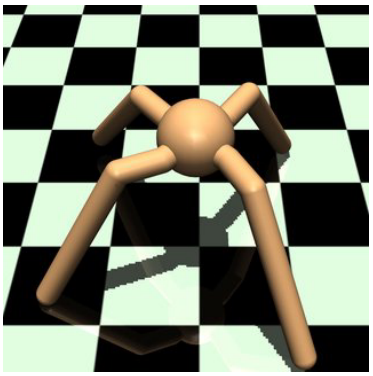
[6] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

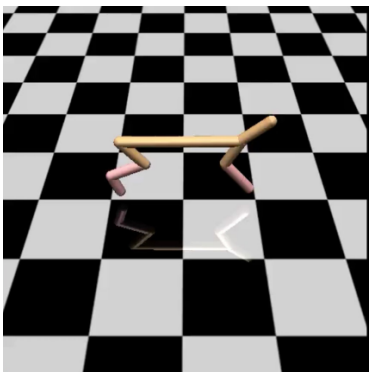
[8] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

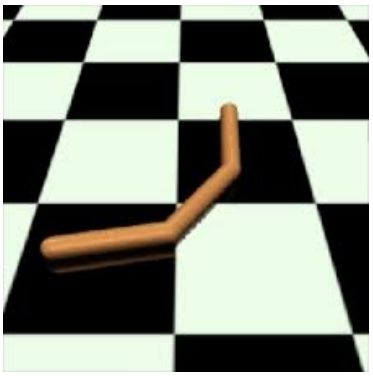
A Gym environments



(a) Ant

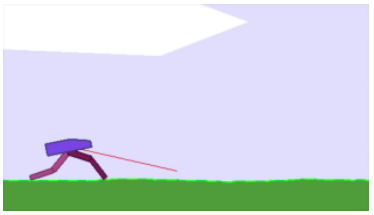


(b) HalfCheetah

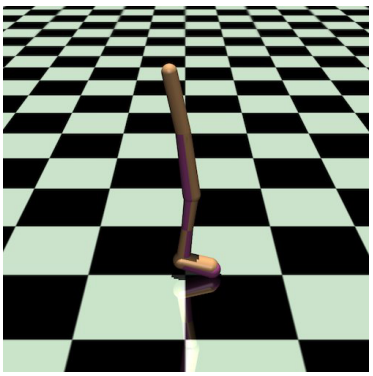


(c) Swimmer

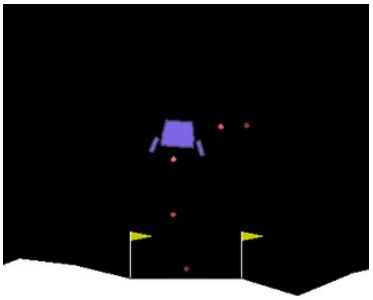
(d) Reacher



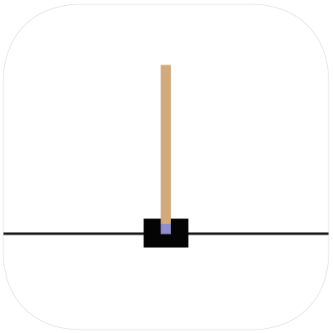
(e) BipedalWalker



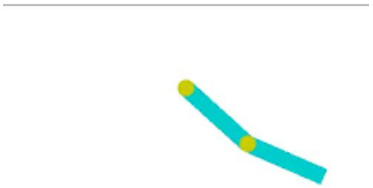
(f) Walkerd2D



(g) LunarLunder



(h) Cartpole



(i) Acrobot

B General results

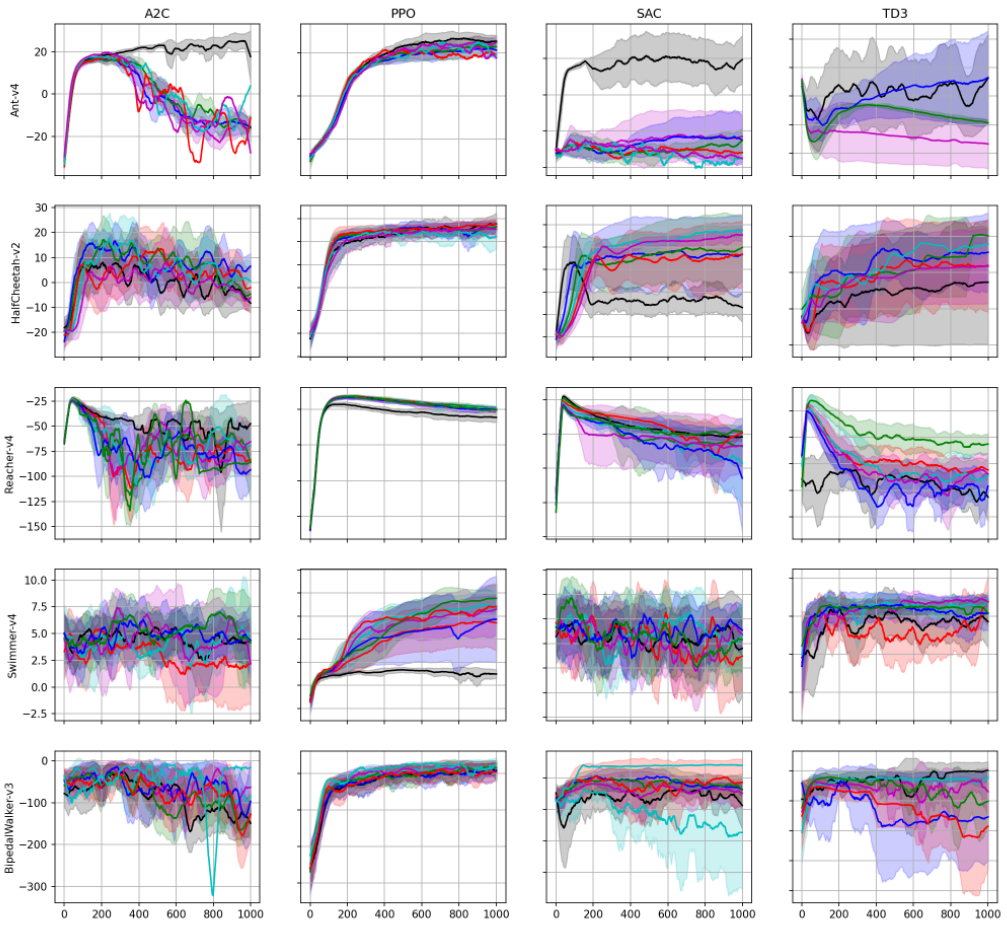


Figure 34: None (black) vs Fourier Light Features, for robotic environments

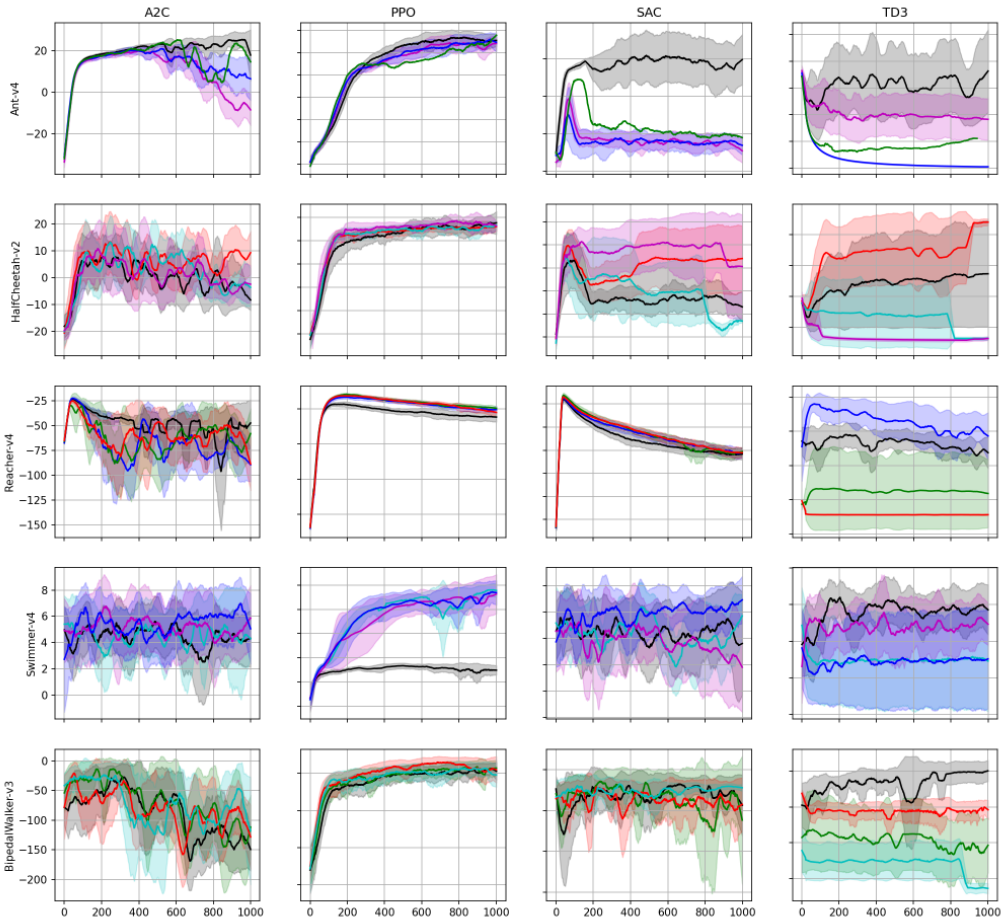


Figure 35: None (black) vs Mix Triangular, for robotic environments

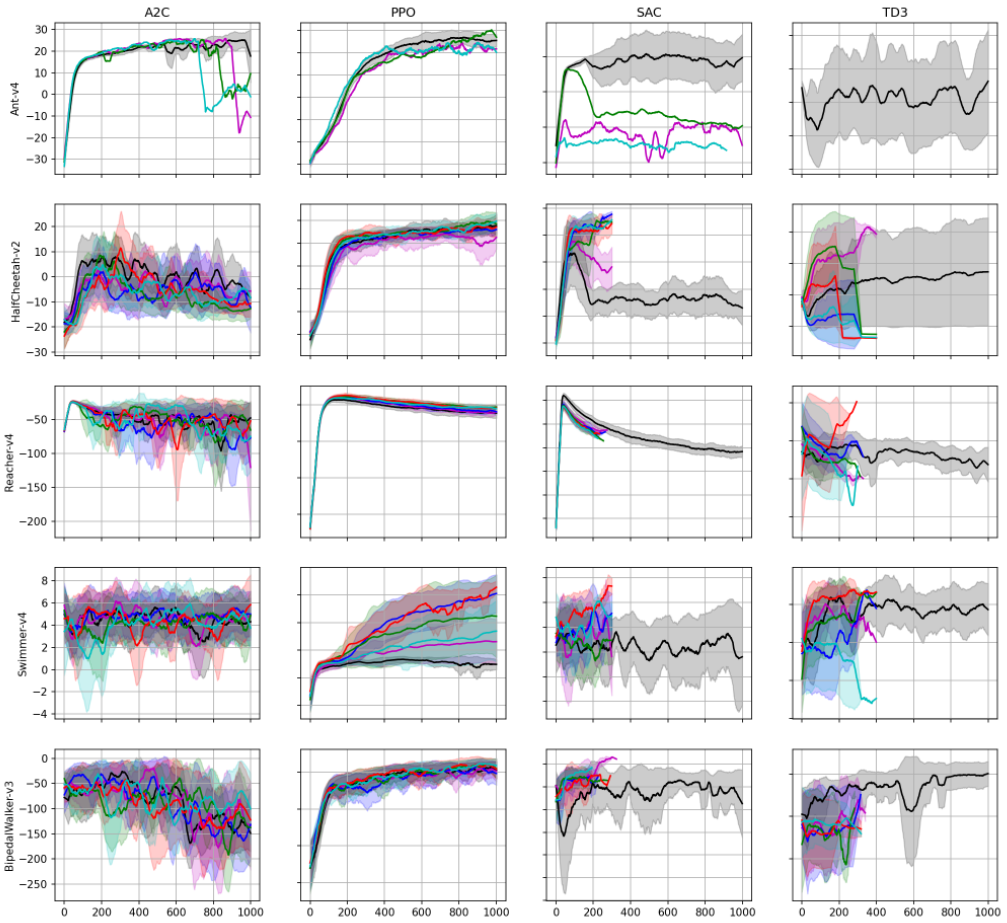


Figure 36: None (black) vs Triangular Light Features, for robotic environments

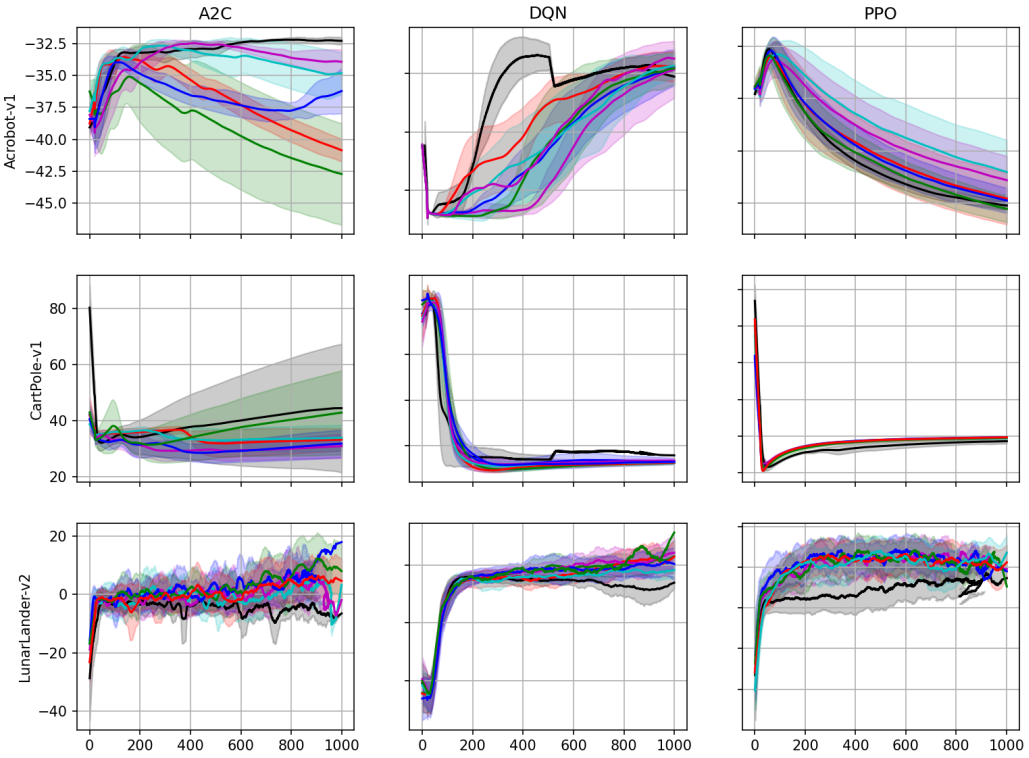


Figure 37: None (black) vs Fourier Light Features, for discrete environments

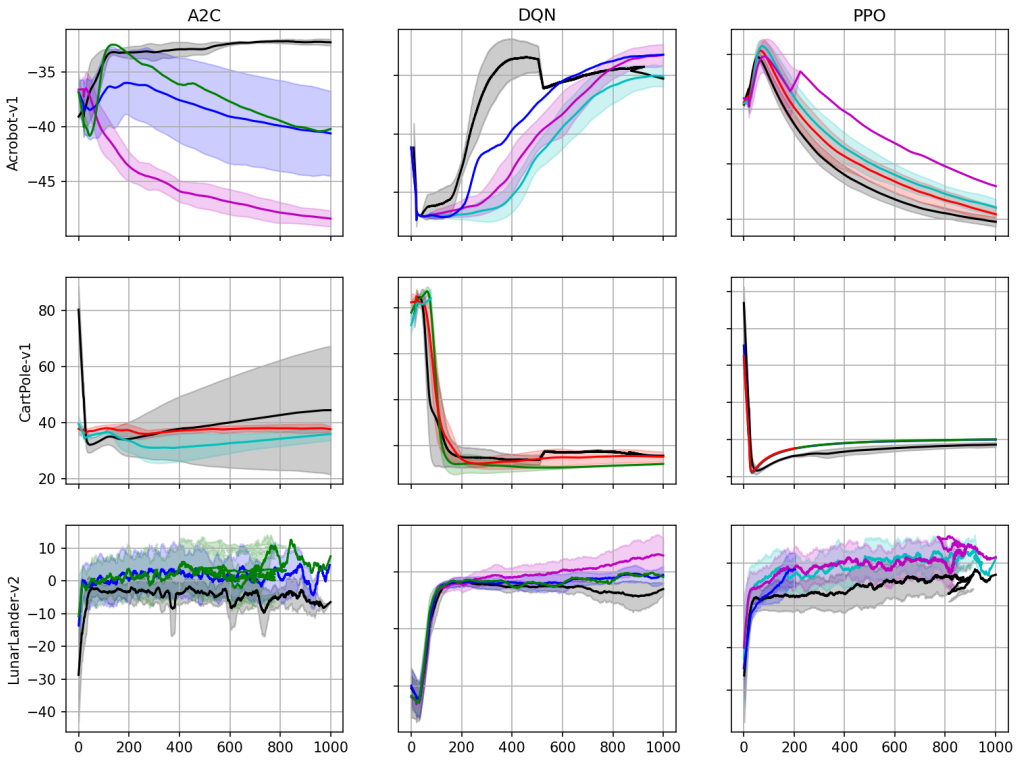


Figure 38: None (black) vs Mix Triangular, for discrete environments

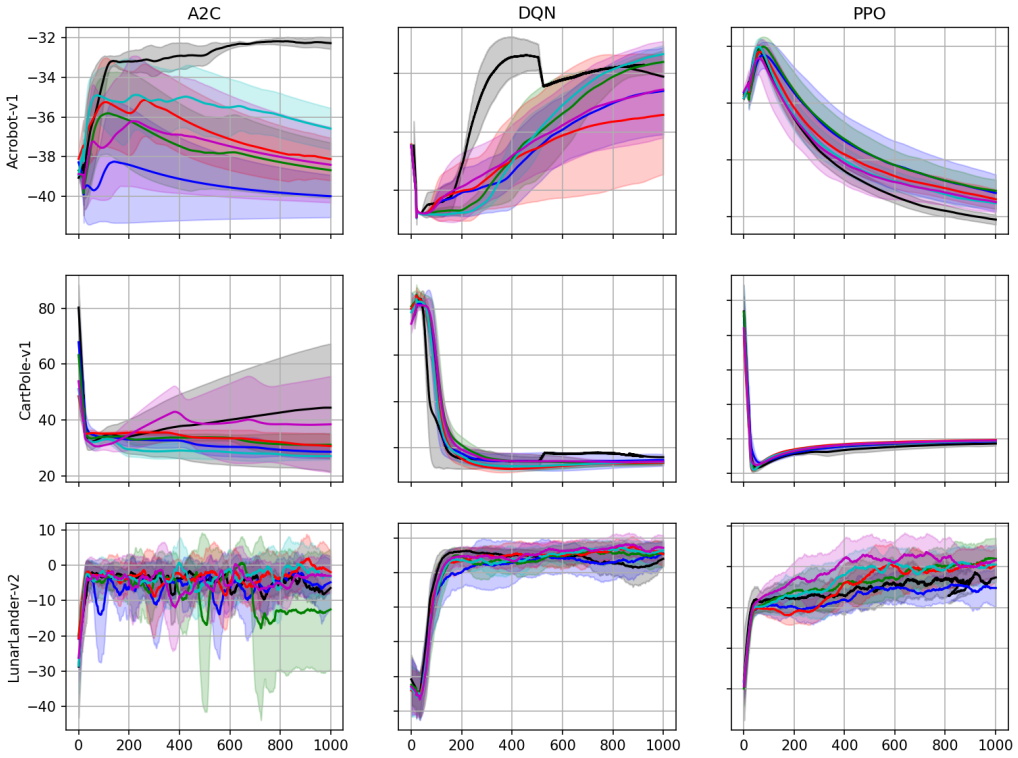
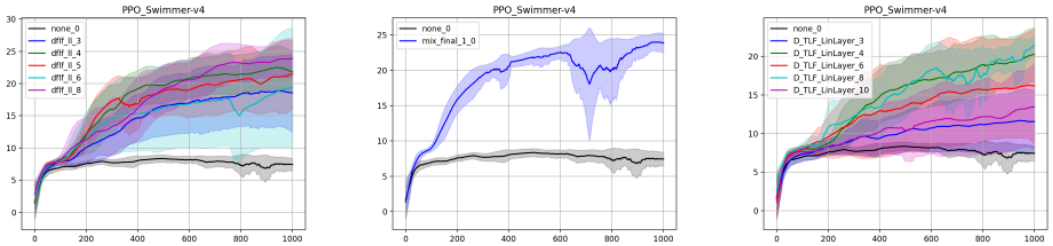
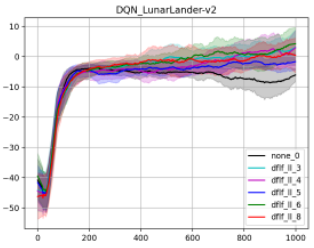


Figure 39: None (black) vs Triangular Light Features, for discrete environments

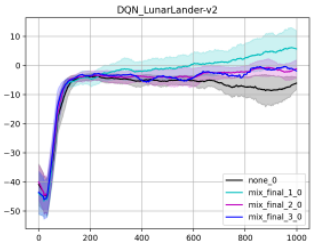
C Specific results



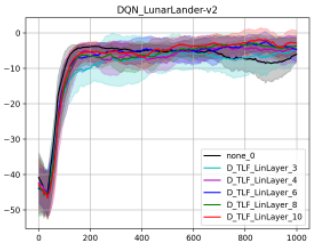
(a) Fourier Light Features (b) Mix Triangular (c) Triangular Light Features



(a) Fourier Light Features



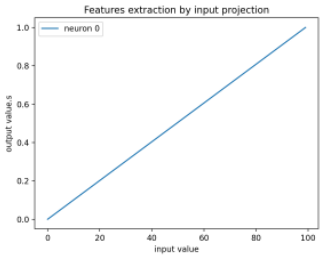
(b) Mix Triangular



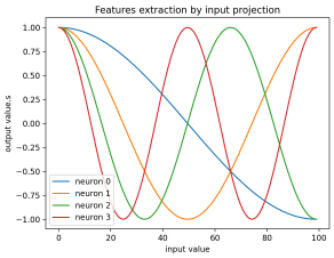
(c) Triangular Light Features

D Layer projection

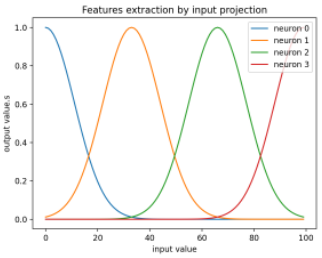
This is all layer projection used in this project :



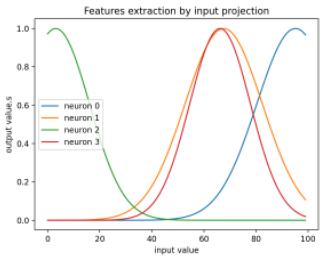
(a) Linear



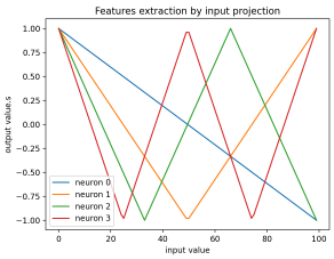
(b) Fourier Light Features



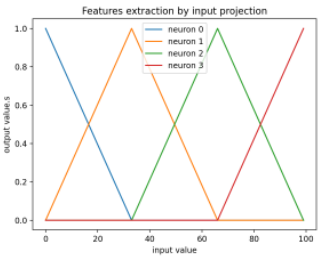
(c) Gaussian fixed



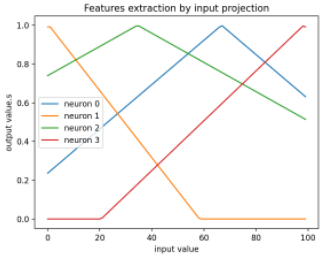
(d) Gaussian learned



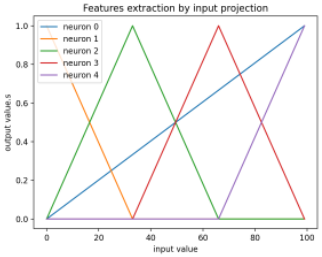
(e) Triangular Light Features



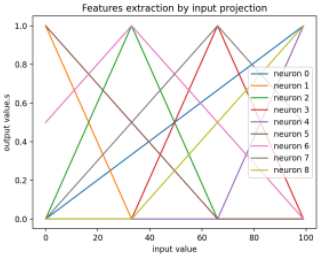
(f) Triangular fixed



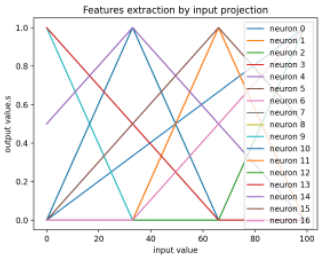
(g) Triangular learned



(h) Mix triangular 1



(i) Mix triangular 2



(j) Mix triangular 3

E Hyperparameters of policies

Based on stable-baselines3 from OpenAI [8], this is our policies with all parameters used :

Table 1: A2C

HP	Value
Neural Network	[Action, Value]
Number of Hidden Layers	[2,2]
Number of Neurons per Hidden Layer	[[64,64],[64,64]]
Activation function	[tanh,tanh]
Discount Factor (gamma)	0.99
Learning rate	0.0007

Table 2

Table 3: DDPG

HP	Value
Neural Network	[Actor, Actor Target, Critic, Critic Target]
Number of Hidden Layers	[2,2,2,2]
Number of Neurons per Hidden Layer	[[400,300],[400,300],[400,300],[400,300]]
Activation function	[ReLU,ReLU,ReLU,ReLU]
Discount Factor (gamma)	0.99
Soft Update Coefficient (tau)	0.005
Learning rate	0.001

Table 4

Table 5: DQN

HP	Value
Neural Network	[Q Net, Q Net Target]
Number of Hidden Layers	[2,2]
Number of Neurons per Hidden Layer	[[64,64],[64,64]]
Activation function	[ReLU,ReLU]
Discount Factor (gamma)	0.99
Soft Update Coefficient (tau)	1
Exploration fraction (epsilon)	auto in [0.1,1]
Learning rate	0.0001

Table 6

Table 7: PPO

HP	Value
Neural Network	[Action, Value]
Number of Hidden Layers	[2,2]
Number of Neurons per Hidden Layer	[[64,64],[64,64]]
Activation function	[tanh,tanh]
Discount Factor (gamma)	0.99
Batch size	64
Learning rate	0.0003

Table 8

Table 9: SAC

HP	Value
Neural Network	[Actor, Critic, Critic Target]
Number of Hidden Layers	[2,2,2]
Number of Neurons per Hidden Layer	[[256,256],[256,256],[256,256]]
Activation function	[ReLU,ReLU,ReLU]
Discount Factor (gamma)	0.99
Soft Update Coefficient (tau)	0.005
Batch size	64
Learning rate	0.0003

Table 10

Table 11: TD3

HP	Value
Neural Network	[Actor, Actor Target, Critic, Critic Target]
Number of Hidden Layers	[2,2,2,2]
Number of Neurons per Hidden Layer	[[400,300],[400,300],[400,300],[400,300]]
Activation function	[ReLU,ReLU,ReLU,ReLU]
Discount Factor (gamma)	0.99
Soft Update Coefficient (tau)	0.005
Learning rate	0.001

Table 12

F Image reconstruction

We detail the image reconstruction task used in this project to represent and understand the input projection.

We have implement an image reconstruction, it is a good way to have a visual understanding of our neural network along dimensions. The purpose of this implementation is not to feat the image, but to show the advantage and limits of each methods with the use of the minimal neurons and layers possible.

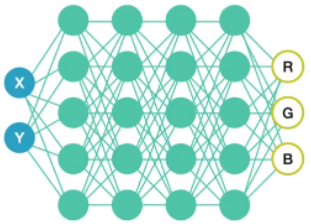


Figure 43: Representation of the neural prediction of the color of a pixel [3]

The task is to find a neural network who will, based on the coordinate of a pixel, to find the color of it.

$$F(x, y) => (R, G, B) \tag{3}$$

F : The function approximation, in our project, a neural network with a specific feature extraction

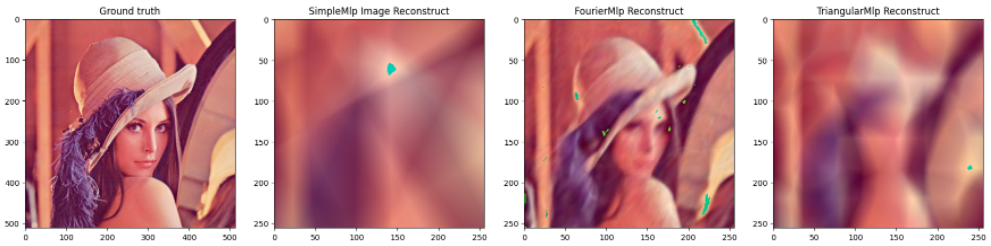


Figure 44: Result of the different methods to

We also can feat any type of signals (images, sound, 3D shape ...), we have choose to feat a 2D sinusoid :

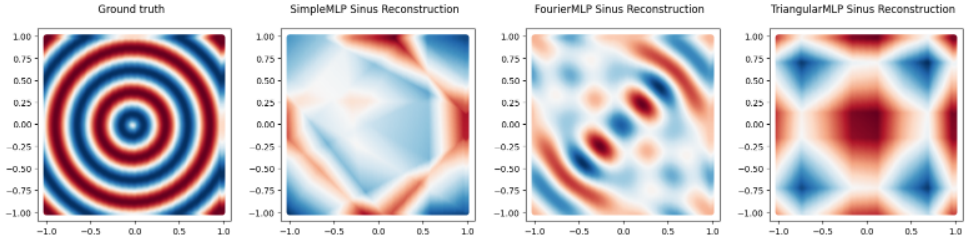


Figure 45: Result of the different methods on 2D Sinus reconstruction

G Fourier Features

In this project, we have used Fourier Features, theorized for the machine learning field by Tancik et al,.[3] .
For our project, we focus on the Determinist Fourier Light Features. This is a little explanation of the different method :

Notation :

$I \in \mathbb{R}^n$: Input matrix of dimension

$X \in [0,1]^n$: Input matrix normalized

$K \in \mathbb{R}^{i,j}$: Features Feature kernel

$H \in \mathbb{R}^{k,l}$: Result of the multiply of the input by the kernel

$Z \in \mathbb{R}^p$: Features Extracted matrix, periodic and flatten version of H

$Y \in \mathbb{R}^a$: Output matrix

$o \in [1, \sim 64]$: order of FFs (by E))

$\mathbb{R}^n \to \omega(X) \to [-1,1]^n$: Periodic method ($\cos(X)$ by **??**)

$\mathbb{R}^n \to \psi \to [0,1]^n$: Normalization method ($[0,1]$ by **??**)

G.0.1 Overview of the global network processing

$I = Input$
$X = Normalize(I)$
$H = X * Kernel * 2\pi$
$Z = FeatureExtraction(H)$

$Y = NeuralNetwork(Z)$

$Output = Y$

The method called $NeuralNetwork$ is the RL agent process, for specific policy, we can have different action architecture, based on the feature extraction (critic/actor neural network for example).

Notation of a classical FF, the normalized input is multiply by the kernel, to be process by a periodic function ($input * kernel = output$) :

$$\omega(\begin{bmatrix} input_{(0)} \\ input_{(1)} \\ ... \\ input_{(n)} \end{bmatrix} * \begin{bmatrix} kernel_{0,0} & kernel_{0,1} & ... & kernel_{0,j} \\ kernel_{1,0} & kernel_{1,1} & ... & kernel_{1,j} \\ ... & ... & ... & ... \\ kernel_{i,0} & kernel_{i,1} & ... & kernel_{i,j} \end{bmatrix}) = \begin{bmatrix} output_0 \\ output_1 \\ ... \\ output_p \end{bmatrix} \tag{4}$$

G.1 Fourier Features / Fourier Light Features

There are two ways to process an Fourier Features extraction, the Fourier Features (FF) and the Fourier Light Features (FLF), all Fourier Features and their variants are note FFs. This technics are taken from (Tancik et al., 2020) and (Brellmann et al., 2021). In this paper, we aim to have a explicit and the simpliest explanation of FFs.

G.1.1 FF : Fourier Feature

This method convert the input to bigger observation space to have more information. With respect of a Fourier Transform, the input are mixed.

In dimensions space, we have : $X^n \rightarrow Z^p$, the p is dependant of the order and of the variant of the FF.

The simplified formula is : $FF_i(X) = \omega((X_0 * K_{0,i} + X_1 * K_{1,i} + ... + X_n * K_{n,i}) * 2\pi)$ with $i \in [0, p]$

The final equation is :

$$Z_i = FF_i(X) = \omega(\psi(I)^\intercal * K * 2\pi) = \sum_{j=0}^{n} \psi(X_j) * K_{j,i} * 2\pi \tag{5}$$

with $shape(K) = (n, p,) \rightarrow K^{n*p}$ we have :

$$\omega(I^{1,n} * K^{n,p} * 2\pi) = \omega(H^{1,p}) = Z^p \tag{6}$$

G.1.2 FLF : Fourier Light Features

This method is a light version of the FF, but with benefits, the output space is proportional to the input space and the order. Each input is decomposed in different periodic reading of the value.

In dimensions space, we have : $X^n \rightarrow Z^p$ with $p = n * order$

The final equation with $i \in [0, n]$ and $j \in [0, order]$ is :

$$FLF_{i,j}(X) = \omega((\psi(I_i) * K_j) * 2\pi) \tag{7}$$

with $dim(K) = (1, o,) \rightarrow K^{1,o}$ we have :

$$\omega(X^{n,1} * K^{1,o} * 2\pi) = \omega(H^{n,o}) = Z^p \tag{8}$$

G.2 Deterministic / Random / Learned Fourier Feature

The main part of the the FFs is the kernel value, they can be determinate, random or learned, this variations cause several changes in the feature extraction processing. The notation is DFF/DFLF for Deterministic Fourier Features or Fourier Ligth Feature, RFF/RFLF for Random FF and FLF, and LFF/LFLF for Learned FF and FLF.

G.3 Deterministic

The Deterministic Fourier Features and the Deterministic Fourier Light Features are input pre processing method who have explicit transformation based on explicit equation witout weight regularization or random value.

G.3.1 DFF

: Deterministic Fourier Features let K be determinate by classical Fourier Features transform. It's a interressant and an explicit technics, each combination of each input is present, but it become impossible when the observation space is too big (Impossible pre-processing : IPP).

With $X^n \rightarrow Z^p, p = o^n$

with $i \in [0, n]$, $j \in [0, p]$ and $dim(K) = (n, p,) \rightarrow K^{n*p}$:

$$K_{i,j} = int(j/o^i) \mod o \tag{9}$$

with int() round to the lower integer

This explain the impossible pre-processing for big input, for example, to resolve the mujoco problem Ant-v4 from [9]

We have as input $n = 27$, we choose a litle order like $o = 2$, we finaly have $p = 2^{27} = 134.217.728$ who is hard to compute and useless to achieve robotics problems.

G.3.2 DFLF

: Deterministic Fourier Light Features let K be determinate by classical simplified Fourier Features coeff

with $i \in [0, o]$ and $dim(K) = (1, o,) \rightarrow K^{1*o}$:

$$K_{1,i} = i + 1 \tag{10}$$

G.4 Random

G.4.1 RFF

: Random Fourier Features let K with random normalized coeff

with $i \in [0, n]$, $j \in [0, p]$ and $dim(K) = (n, p,) \rightarrow K^{n*p}$:

$$K_{i,j} = r \in [0, 1] \tag{11}$$

G.4.2 RFLF

: Random Fourier Light Features let K with random normalized coeff

with $i \in [1, order + 1]$ and $dim(K) = (o,) \rightarrow K^o$:

$$K_i = r \in [0, 1] \tag{12}$$

G.5 Learned

LFF and LFLF, Learned Fourier Features and Learned Fourier Light Features let K be find by the neural network by basic SGD. The LFF and LFLF is close to the SIREN (Sitzmann et al., 2020 [6]), a neural network with sinusoidal activation function.

G.5.1 LFF

: represented by : $SIREN(X) = \omega(X * K)$ X*K is classical full connected layer :

with $dim(K) = (n, p) \rightarrow K^{n*p}$:

$$\omega(X^{1*n} * K^{n*p} * 2\pi) = \omega(H^{1*p}) = Z^p \tag{13}$$

G.5.2 LFLF

: represented by : $SIREN_L(X) = \omega(X * K)$ X*K is classical full connected layer :
with $dim(K) = (1, o) \rightarrow K_{1*o}$:

$$\omega(X^{n*1} * K^{1*o} * 2\pi) = \omega(H^{n*o}) = Z^p \tag{14}$$