Exploring the Potential of the Forward-Forward Algorithm in Deep Reinforcement Learning

Robin Junod
Supervisor: Giulio Romanelli
Swiss Data Center, EPFL, Switzerland

July 6, 2025

https://github.com/RobinJunod/FF_DRL

Abstract

Deep learning was created as an attempt to replicate the workings of biological neurons. The initial goal was to create AI systems that functioned similarly to the human brain. Despite our partial understanding of how neurons activate and communicate, the actual learning process in a biological brain remains a mystery. In response to this challenge, the backpropagation (BP) algorithm was invented. This algorithm computes a gradient using the derivative's chains rule and updates neural network weights with gradient descent. Although backpropgation has proven to be a great method to make a neural network learn, it seems implausible that a biological brains learn that way.

The Forward-Forward (FF) algorithm uses a new method to train a neural network and was introduced by Geoffrey Hinton [1]. Instead of using the traditional forward and backward passes for backpropagation, it relies on two forward passes: one with real data (positive) and another with data generated by the network itself (negative). Each layer aims to maximize its activity for positive data and minimize it for negative data. This new algorithm has shown good results for classification tasks but has never been implemented in reinforcement learning (RL). Our objective is to investigate the algorithm's potential within the field of reinforcement learning.

1 Introduction

To initiate this project and explore the application of this new algorithm in the context of deep reinforcement learning, we have chosen to tackle the widely recognized CartPole [2] problem as our initial task. It consists of a cart that can move left or right along a track, with a pole attached to it. The goal is to balance the pole on the cart for as long as possible. This problem has been chosen due to its simplicity and popularity as it is often used as a benchmark to compare reinforcement learning (RL) algorithms. Despite it being really easy to solve using conventional Deep Q-learning (DQL) approach, using the Forward-Forward algorithm (FF) for this task is very challenging. The main reason is that the forward forward is not well suited to solve regression tasks, which is a critical aspect for reinforcement learning algorithms such as DQL. In this report, three distinct approaches have been introduced and tested to adapt the FF algorithm to RL:

- The Survival-Focused algorithm
- The Deep Q-learning algorithm
- An advanced Deep Q-Network a using convolutional neural network (CNN)

The first two algorithms were executed in the CartPole-v1 environment, while the third algorithm was tested in the more complex Breakout-NoFrameskip-v4 environment. This approach enables us to address environments where the state is represented either as a vector or as an image.

1.1 The Forward-Forward algorithm, an alternative to backpropagation

This idea has been introduced first in 2022 by G. Hinton [1]. The main motivation for such an algorithm comes from the fact that the backpropagation is not biologically plausible. We don't have strong proof that the cortex directly conveys error derivatives or holds the neural activities for later use in a backward pass. Moreover, the main computational cost in deep learning comes from this backward propagation. Finding new algorithms improve the learning process of neural nets, could be a revolution in the field of deep learning. The training cost is currently enormous for large language models (LLMs). For instance, 1,287 MWh were used in ChatGPT-3's training phase [3] which corresponds to millions of dollars, and a huge environmental impact.

The Forward-Forward tries to solve these issues and proposes a new way to make a neural net learn. It works as follows; It replaces the forward and backward passes of backpropagation by two forward passes, one with positive (i.e. real) data and the other with negative data which could be generated by the network itself. Each layer has its own objective function which is simply to have high goodness for positive data and low goodness for negative data [1]. The goodness is the square sum of each hidden units 1. One more important thing to add is a normalization at the beginning of every layers to remove any trace of goodness. Without this normalization step, if the first layer efficiently discriminates positive and negative data, the second layer could simply replicate the same vector as an output. The normalization process eliminates information about the norm of the first layer's output, compelling the second layer to learn positive and negative data from a normalized vector.

1.1.1 The Goodness

The goodness is a concept introduced by G.Hinton[1]. It is a simple way to determine whether a neuron has high or low activation and is defined as follow :

$$\text{Goodness} = \sum_j y_j^2 \tag{1}$$

where y_j is the activity of hidden unit j before layer normalization. The probability that a sample is positive is then defined as such :

$$p(\text{ positive }) = \sigma\left(\sum_j y_j^2 - \theta\right)$$

Where σ is the sigmoid function and θ the threshold. The threshold serves to distinguish between positive and negative data. When the goodness surpasses θ, we consider the input as positive; otherwise, it's considered negative.

1.1.2 The Loss function

When dealing with the Forward-Forward, the loss function is computed at each layer. This loss function should have the property of increasing the activation for positive data while decreasing it for negative data. There are various types of loss functions that can serve this purpose, but we chose the following one due to its stability and performance:

$$\text{Loss } = \frac{1}{N}\sum_{i=1}^{N} F\left(-g_{\text{pos }i} + \theta\right) + \frac{1}{N}\sum_{i=1}^{N} F\left(g_{\text{neg }i} - \theta\right)$$

Where:

- N is the number of samples or data sample.
- $F(x)$ is the softplus function, defined as $F(x) = \log(1 + \exp(x))$

2 Survival-Focused Reinforcement Learning

The initial idea to adapt the FF algorithm to reinforcement learning was to discriminate positive and negative data by determining positive data as the state-action pair that increases the future reward, and negative data as being the state-action pair that decreases the future reward. A network that would be trained with this idea will have high goodness for actions that leads to high rewards and a low goodness for action that are not going to get a lot of rewards.

With this concept in mind, the objective is to automatically classify actions that result in high rewards. This task poses a challenge because determining good and bad actions without relying on the traditional Bellman equation is not straightforward. Our initial approach involves the following strategy: We store the state-action pairs collected in the course of a complete episode. Once the episode is over, we define the 5-20 state-action pairs that precede the agent's death as negative data, and the others action as positive data (see Section 2.1).

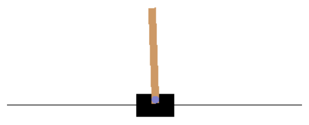


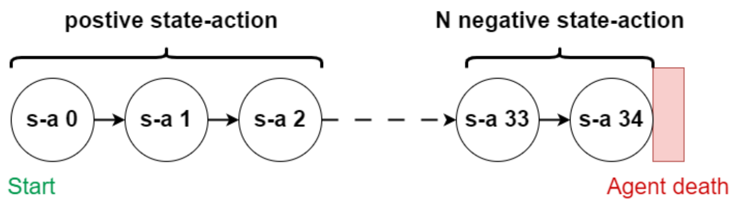
Figure 1: Cartpole [2]

Algorithm 1 Survival-focused

```
1:  Input: Learning rate $\eta$, variance control parameter $\lambda_t$
2:  Initialize: replay memory , FF network,
3:  for all episodes do
4:      Reset state
5:      repeat
6:          if epsilon greedy then
7:              $action$ = random
8:          else
9:              $action$ = choose action with highest goodness
10:         Play action
11:         Store state-action pair in replay memory
12:     until Death
13:     Negative data: Add $N$ state-action pairs before agent's death
14:     Positive data: Add remaining state-action pairs from replay memory
15:     Train the FF network with positive and negative data
```

2.1 Data collection for the Survival-Focused Algorithm

This algorithm deviates from the typical DQN approach. Instead of taking only a state as input, the neural network is fed a state-action pair. This pair is formed by one-hot encoding the action and concatenating it with the state. The FF neural network processes the state-action pair, aiming to maximize goodness for positive data and minimize it for negative data. In this context, positive data are state-action pairs that increase the agent's survival probability, while negative data consist of state-action pairs considered bad for the agent survival.



The FF network chooses the best action by trying all possible actions for a given state and selecting the action with the highest associated goodness. Additionally, an epsilon-greedy strategy is employed to balance exploration and exploitation. After the episode concludes, the network transitions into its learning phase, utilizing the collected state-action pairs. Negative data comprise the state-action pairs leading up to the death of the agent, while positive data include the remaining state-action pairs.

2.2 Model

The neural network utilized in this study is a simple multilayer perceptron (MLP) consisting of three hidden layers with respective neuron counts of (50, 20, 20). The network's input comprises a state-action pair (In the case of Cartpole, the input is 4 dimension for the state plus 1 dimension for the action so 5 dimensions in total) (see Figure 2)..

Unlike conventional neural networks that generate standard outputs, this MLP is designed with another objective. It doesn't have an output layer. Instead, it provides the goodness value, which is the strength of the hidden unit activation (see Equation 1). It acts as a classifier, its goal is to categorize input data into either positive or negative data. High activation of network units (where goodness ¿ θ) signifies positive data, while weak activation (where goodness ¡ θ) signifies negative data.

During the inference process, the model evaluates all possible actions for a given state (for n actions it will create n state-action pairs). The model then selects the state-action pair with the highest goodness, and the associated action is played. In essence, the network functions as a classifier, determining the quality of state-action pairs.

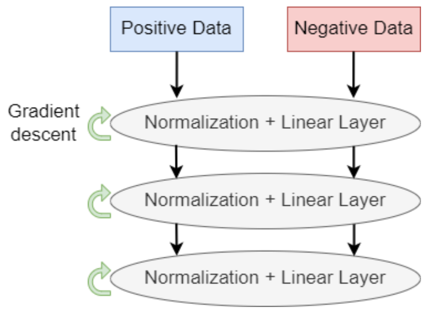

Positive Data
Negative Data
Gradient descent
Normalization + Linear Layer
Normalization + Linear Layer
Normalization + Linear Layer

Figure 2: Network model

2.3 Results and discussion

Adjusting several hyperparameters was attempted to potentially enhance the results, yet surprisingly, these adjustments did not yield a significant impact. The hyperparameters subjects to experimentation included:

- Number of episodes
- Number of state-action before death defined as negative data (horizon death)
- Increasing vs decreasing horizon death during training
- Replay memory size

Despite the effort to fine-tune these parameters, their influence on the overall outcomes appeared to be rather limited.

Overall performence results of the Survival-Focused method Comparing this first implementation with a random agent is a first interesting step to take. This proves us that the network is indeed learning something even and that there can be room for improvement in this adaptation of the FF in reinforcement learning. This first plot proves us that the agent is performing significantly better than a random agent.

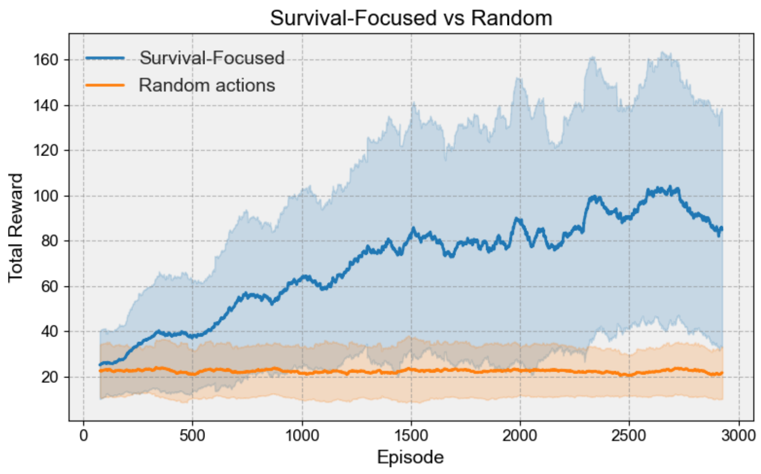


Figure 3: FF survival focused comparison with random actions

Comparison of training methods When it comes to training a network using the forward pass, two distinct approaches can be considered. The first approach involves training the network on a layer-by-layer basis. In this scenario, all the data is passed through the first layer for a specific number of epochs until that layer reaches a satisfactory level of training. Then, the second layer goes through a similar training process, utilizing the output from the first layer as input etc. This training continues until all layers within the network have been trained. The second approach is to train all layers simultaneously by processing one batch of data after another through the network. Even if the first method seems to be more stable, it is certainly less biologically plausible.

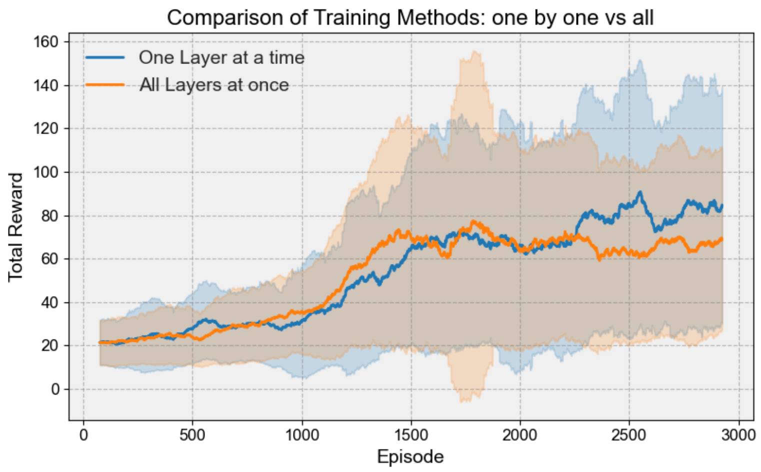


Figure 4: Training method comparison

As shown by the graph (see Figure **??**), the differences between these two types of training are not significant enough to conclude that one is better than the other.

Negative data range tuning The negative data range (or 'death horizon') denotes the count of state-action pairs labeled as negative, specifically the N state-action pairs preceding the agent's death (see Section 2.1). Fine-tuning this number is really important and the first observation that was made is that N should fall within the range of 3 to 30. Ranges exceeding 30 or going below 3 prevent the network from learning.

To optimize results, an interesting improvement is to modify the number N (indicating state-action pairs defined as negative) during the training process. For instance the first experiment, shown by the blue line in (see Figure 5), was to start with a N=5 and increasing it linearly until 10 during the first half of the training (the first 1'500 episodes).

The results shown in (see Figure 5) indicates that the ranges between 5 and 20 can be utilized to have optimal results. Using N higher than 20 will lead to inefficient learning exceptionally in constant range. Also we can notice that decreasing horizon seems to have slightly better results than increasing it.

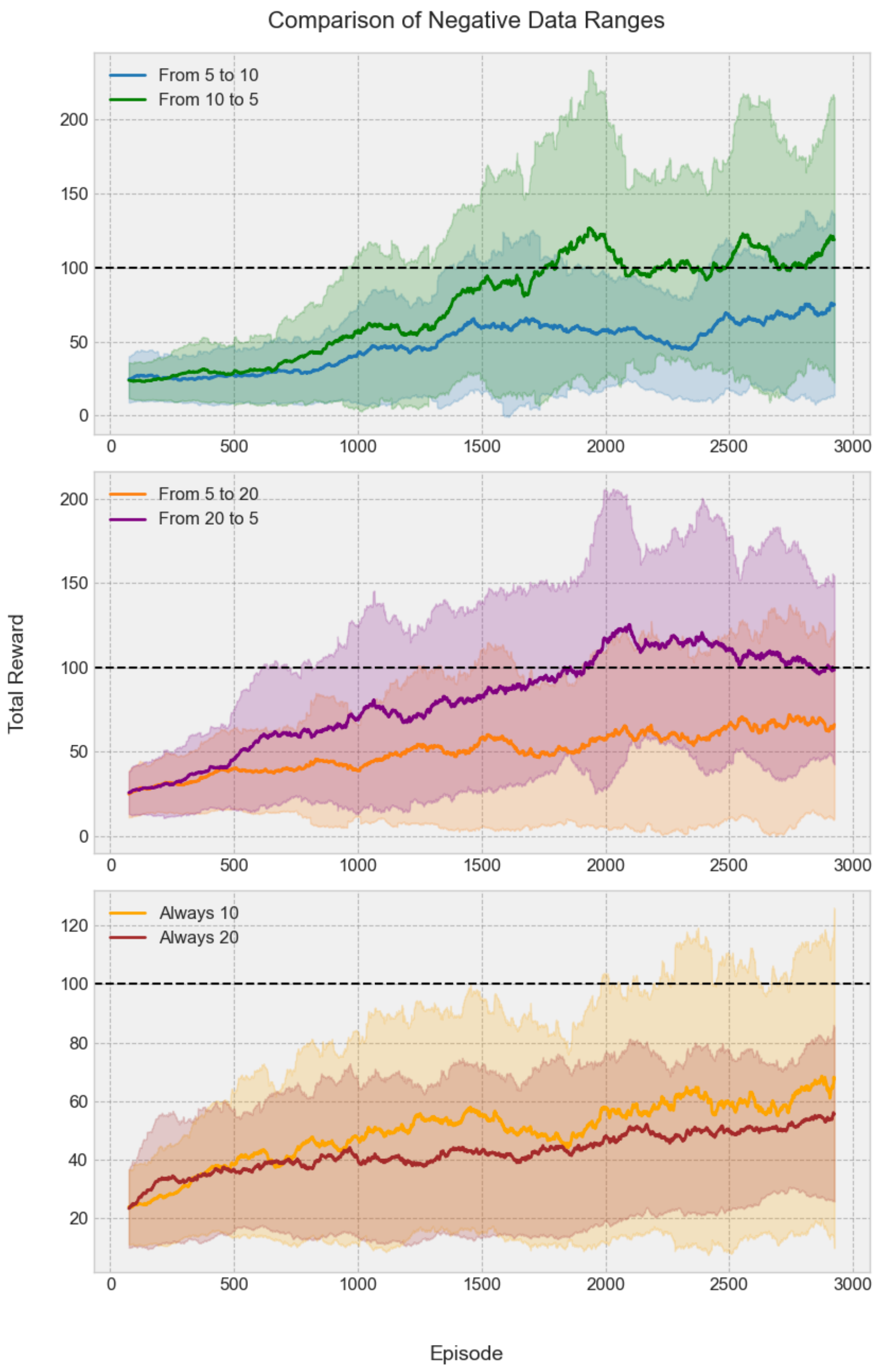


Figure 5: Horizon death range comparison

It's worth noting that reducing the range of negative data during training appears to lead to improved performance (see Figure 5). Decreasing it from 10 to 5 or from 20 to 5 produces similar results, achieving total rewards exceeding 100. This performance is significantly better than what a random agent can achieve.

2.4 Conclusion

Although this approach appears simple, it has shown promising results in the cartpole environment, indicating that Forward-Forward (FF) does exhibit some learning capabilities. However, it falls short when compared to backpropagation methods and is unable to fully solve the cartpole problem, performing at best between 80-100 average reward.

The fundamental limitation of this approach lies in the necessity to predefine what constitutes a good state-action pair, and the model does not learn a Q-function. Furthermore, the primary objective of maximizing the agent's survival time may not be suitable for addressing all types of reinforcement learning problems/environment.

3 Deep Q-Learning with forward forward

Q-learning is a very popular method in reinforcement learning. The Q-value, often denoted as "Q(s, a)", is a function estimating the expected discounted cumulative reward an agent can achieve by taking a specific action "a" in a particular state "s" and then following its current policy thereafter. It's important to note that the policy to follow afterward is not necessarily an optimal one; it is the current policy. The Q-value is also a function of the policy, as the action with highest Q-value is chosen when it. Its goal is to help the agent make informed decisions to maximize its long-term rewards. In Deep Q-learning (DQL), a neural network is used as a function approximate for the Q function.

This neural network solves a regression task. A Deep-Q Network (DQN) takes as input the current state "s" and outputs every Q-values for that state 's', corresponding to all possible actions that can be taken (e.g. with 3 possible actions, the output will be : Q(s,a1), Q(s,a2), Q(s,a3)). The equation below is part of Q-learning and is called the Bellman Equation. This foundational result in RL helps update the Q-values by considering immediate and future rewards.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_a Q\left(s', a\right) - Q(s,a)\right]$$

In this equation:

- $Q(s,a)$ is the Q-value for taking action 'a' in state ' s '.
- α is the learning rate, which determines how quickly the Q-values are updated.
- R is the immediate reward received after taking action ' a ' in state ' s '.
- γ is the discount factor, which weighs the importance of future rewards.
- $Q\left(s', a\right)$ is the Q-value for the next state 's' and the optimal action 'a' in that state.
- The update rule adjusts the Q-value for the current state-action pair 's, a' based on the received reward and the expected future reward.

The real challenge to adapt this Q-learning method to the FF algorithm is to be able to use the FF to solve a regression problem which, to the best of our knowledge, has not been implemented in the past. FF focuses on the fact that there is positive and negative data, this algorithm is well suited for classification task as the label can be directly encoded into the input data. For instance, in the original paper [1] (MNIST case), the network's input is a combination of an image and its corresponding (one-hot encoded) label. Negative data is defined as images paired with incorrect labels, while positive data consists of images paired with the correct labels. During the inference phase, the network concatenates each potential label with the given image, and the input with the highest goodness is presumed to be the correct one.

However, this approach is not applicable to regression tasks due to a fundamental limitation. Specifically, it is impossible to one-hot encode a target value (float) for regression. Attempting to consider every possible float value as input during inference would entail an infinite number of steps, making this methodology impossible for regression tasks.

3.1 Regression with Forward-Forward

The first step is to create a new algorithm that could solve a regression task with the Forward-Forward. Our approach to solve a regression task with the Forward-Forward (FF) method involves utilizing the network as a feature extractor. The idea is to extract features from the FF's output and subsequently employ these features as input for a linear model. The scheme below gives an idea of how the structure of the model will work:
The primary challenge lies in the generation and differentiation of positive and negative data.

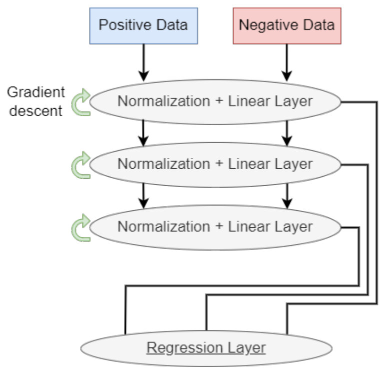

Positive Data
Negative Data
Gradient descent
Normalization + Linear Layer
Normalization + Linear Layer
Normalization + Linear Layer
Regression Layer

Figure 6: Network DQN with Forward-Forward

3.1.1 The creation of negative data

Creating negative data for the Forward-Forward method to address regression tasks pose a significant challenge. Geoffrey Hinton proposed an approach for generating negative data in the context of images, involving filters that maintains micro-level coherence while introducing significant macro-level differences from real data.

In our implementation, we aim to generate negative data that would highlight some crucial features. We introduce a method for generating negative data, but future research could be to explore alternative approaches for negative data generation.

Data Shuffling Our proposed approach, referred to as "data shuffling," involves randomizing the data across all dimensions. For instance, to generate a new negative sample, the procedure selects a random value from dimension 1 of the entire dataset, followed by a random value from dimension 2, and so on. This type of shuffling aims to eliminate any correlations between different dimensions. This is particularly interesting as the Forward-Forward (FF) can learn to distinguish between data with and without such correlations. Additionally, it is worth noting that this concept bears similarities to permutation importance, a well-established concept in the field. Below is an illustration of such a procedure.

$$\begin{bmatrix}1\\2\\3\end{bmatrix}, \quad \begin{bmatrix}4\\5\\6\end{bmatrix}, \quad \begin{bmatrix}7\\8\\9\end{bmatrix} \rightarrow \quad \begin{bmatrix}4\\2\\3\end{bmatrix}$$

3.1.2 Dataset

To evaluate our new regression model using the Forward-Forward, our goal is to create a dataset with correlations among its dimensions. This correlation is crucial as we employ the "data shuffling" method to generate negative data. Moreover as the goal is to deploy it on the Cartpole environment, we generated a dataset with three Gaussian distributions in 4 dimensions (reflecting the 4 dimensions states in CartPole). The visualizations of this dataset from various dimensional view are presented below.

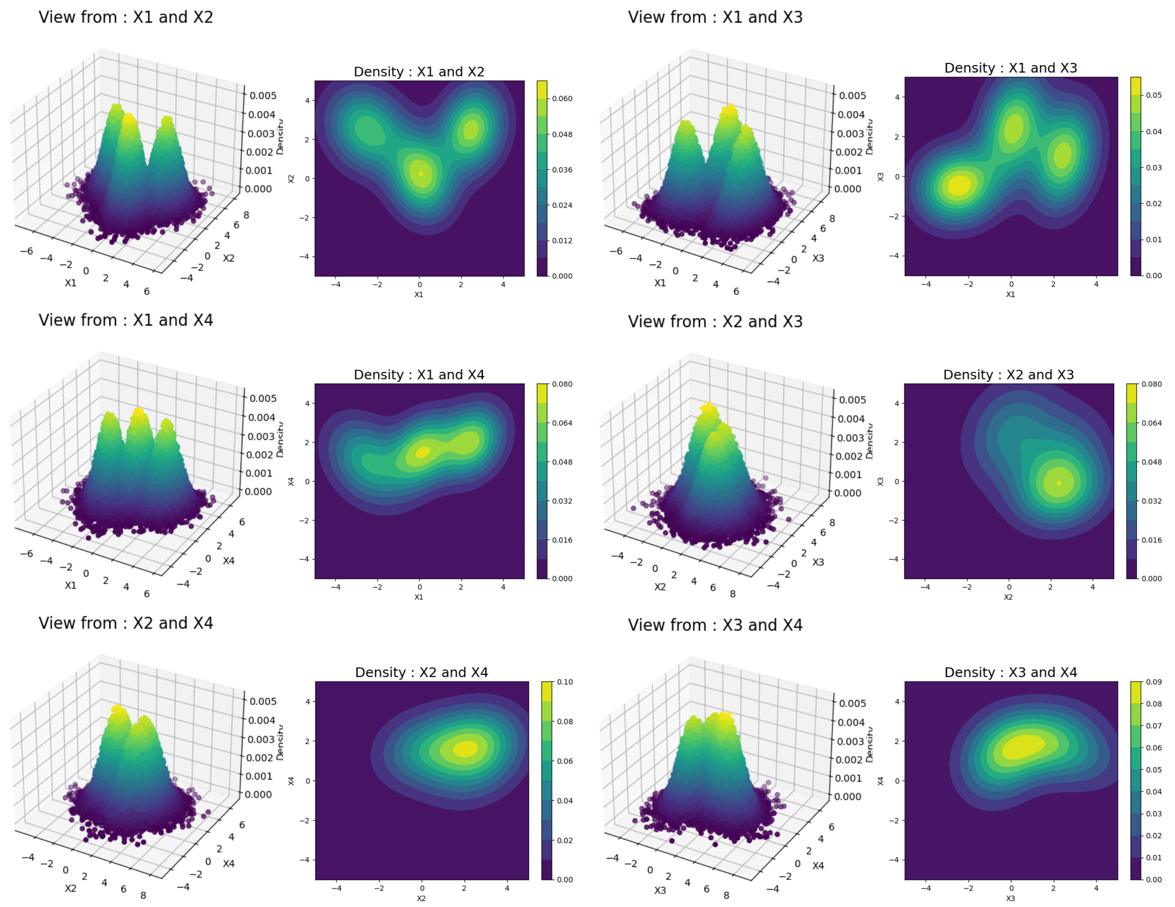


Figure 7: Dataset viewed from different dimensions

3.1.3 Model

The utilized network comprises two components: a feature extractor and a regression layer (linear layer). The feature extractor is constructed using a Multi-Layer Perceptron (MLP) with two hidden layers sized as follows: [input size (4), 10, 5]. In the regression component, a single linear layer is employed. The input is the features given by the first component and the output should fit the target defined below.

Non-Linear Target The non-linear target for this regression task has to have some form of

$$Target = 2x_1 + x_2^2 + x_3x_4$$

Loss function for the Forward-Forward The choice of a suitable loss function is crucial for maximizing the model's performance on positive data while minimizing it on negative data. To address this challenge, two different loss functions were explored.

The first, simpler approach defines the loss as a linear function:

$$Loss = NegativeGoodness - PositiveGoodness$$

This formulation encourages an increase in the goodness of positive data and a decrease in the goodness of negative data to minimize the loss. However, this loss function is not stable enough for training the model over a large number of epochs. It leads to constant weight updates, eventually causing divergence. The

To overcome this issue, the softplus loss function, illustrated here 2, was adopted.

Algorithm 2 Loss function

```
1: positive loss = mean(softplus(-goodness positive + threshold))
2: negative loss = mean(softplus(goodness negative - threshold))
3: loss = positive loss + negative loss
```

3.1.4 Results, FF regression on custom dataset

To evaluate the performance of the new algorithm, we conducted 20 complete training (of 100 epochs) to fit the dataset using Forward-Forward (FF) and backpropagation. The results illustrate the progression of the R2 score, showcasing both the average and standard deviation across these 20 training iterations. Based on these outcomes, we notice that the FF method is not distinguishable from backpropagation in the initial 20 epochs, but is then completely outperformed by the backpropagation latter on. The forward forward is far from surpassing backpropagation but these early results are promising.

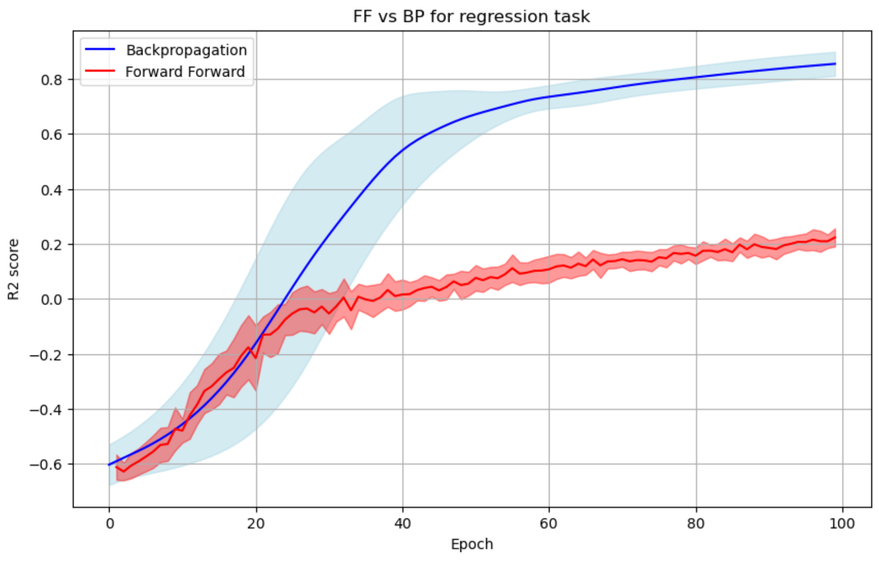


Figure 8: Average Evolution of R2 Score Across 20 Complete Training Sessions

Future work We only used the data shuffling method to get these results. To improve the current model, it's worth investigating different methods of generating negative data and possibly combining these approaches. The generation of negative data is the key for effective learning with the Forward-Forward.

3.2 Q-learning with Forward-Forward

Having identified a methodology that allows a neural network to learn a regression task using Forward-Forward, the current objective is to apply this approach to train the Deep Q-Learning (DQL) algorithm with FF. The overall process follows the concept illustrated in Figure 6. The learning procedure is made of two distinct steps.

First Step: Train the Feature Extractor with FF In this phase, a series of random actions is played to generate real states. These randomly generated states are defined as positive data. To create negative data, the shuffling technique is employed, effectively removing correlations between dimensions. By utilizing both positive and negative samples, the feature extractor is trained with the FF algorithm.

Second Step: Train the Last Layer with DQL The second step is to train the last layer. It utilizes the trained feature extractor to obtain features from the state. With these features, the DQN algorithm is employed to train the last regression layer. The network takes the state as input, passes it through the trained feature extractor, and then processes these features through the last layer, which outputs Q-values like a classical DQN. Notably, the feature extractor remains unaltered during this step.

Repeat Multiple Full Training Using the experiences collected during the second step, the feature extractor undergoes a Forward-Forward training with these new states/data. This iterative process should enhances the learning capabilities of the feature extractor.

Algorithm 3 Deep Reinforcement Learning Model using Q-learning

```
 1: Initialize feature extractor
 2: Add trainable linear layer on top of the feature extractor
 3: Initialize experience replay memory
 4: for $i = 1$ to numberOfFullTraining (10) do
 5:     Generate negative data from experience replay
 6:     Train the feature extractor with Forward-Forward (and data from experience replay)
 7:     Define: DQN model combining feature extractor and linear layer
 8:     while training of last layer not converged do
 9:         Generate experience using DQN
10:         Store experience in replay memory
11:         Sample mini-batch from replay memory
12:         Update only the final layer of the DQN
13:         Occasionally update target layer weights
14: End
```

3.2.1 Last Regression Layer and OLS Limitations

Even if the last layer consists of a simple regression layer with a Mean Squared Error (MSE) loss, Ordinary Least Squares (OLS) can't be used. For the last regression layer, we need to have something that is trainable in reinforcement learning. The main drawback with OLS is that it can't be used in a DQN due to the fact that DQN doesn't optimize all the weight at the same time. It will optimize over the Q-value chosen by the algorithm, which corresponds to only one node of the output dimension. For this reason, the last regression layer was trained using simple gradient descent.

3.2.2 Results and Discussion

The outcomes achieved with this new approach are significantly more promising compared to the previous survival-focused method. Notably, the new method successfully solved Cartpole in certain episodes (within a 500-episode limit), a feat previously unattainable with the survival-focused method. However, when comparing this new method to the standard DQN with backpropagation, it becomes apparent that the learning process is less stable.

These results are indeed very promising and show significant improvement compared to the Survival-Focused method. The only issue arises from the fact that the learning process is less predictable than that of a traditional DQN with backpropagation.

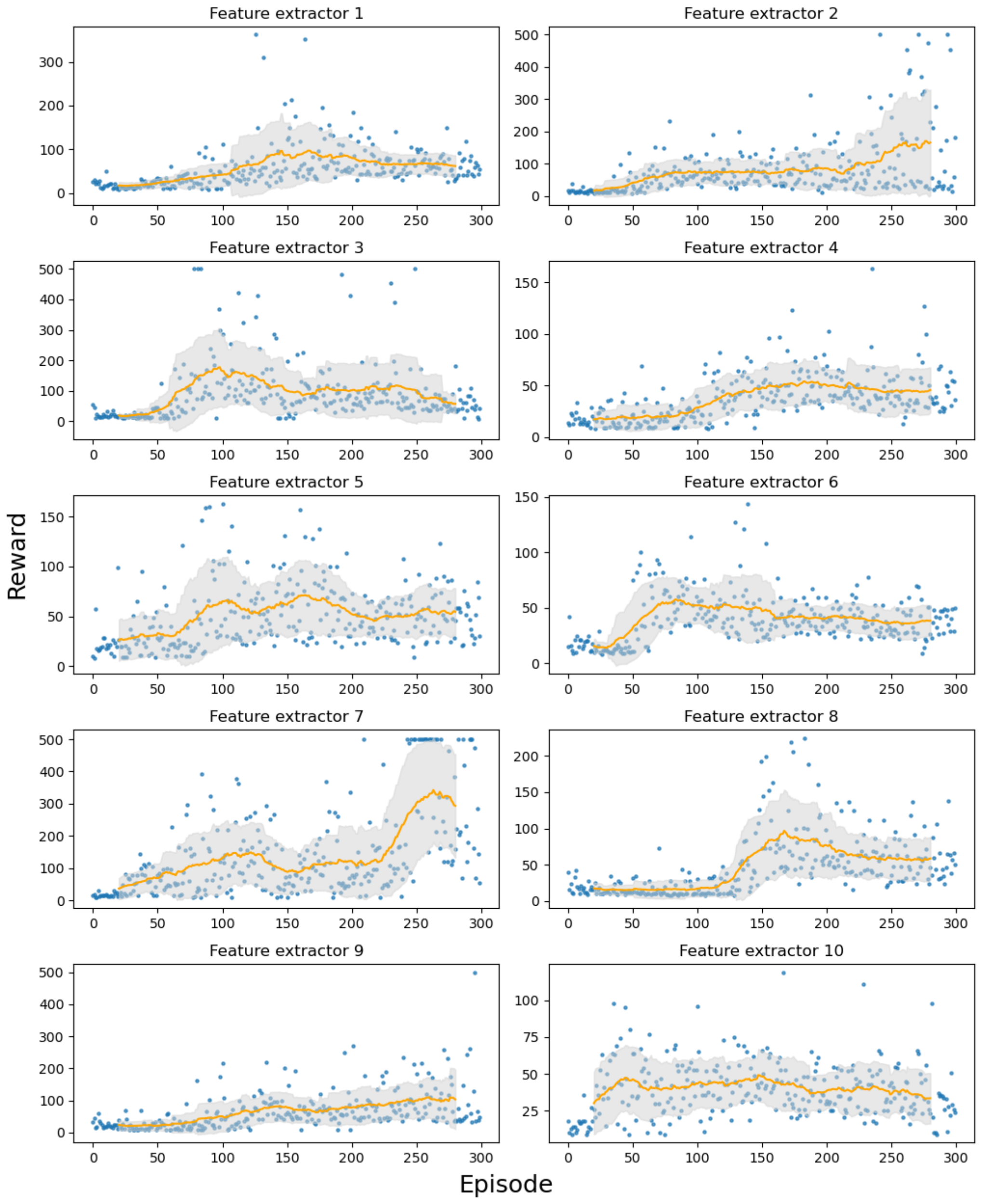


Figure 9: Reward Evolution of the DQN trained with FF

In certain cases, it didn't lead to optimal performance, while others exhibited surprisingly good results. For example, the feature extractor 7 demonstrated exceptional performance, nearly solving the Cartpole environment. When evaluated through inference (excluding the epsilon-greedy exploration), it consistently solved Cartpole in over 90% of the trials.

4 Forward-Forward in Complex Environment

The current objective is to extend the Forward-Forward algorithm to more complex environments where the state is represented by images. Atari games, available in the Gym environment [2], provide an ideal setup for this goal. Given that the data consists solely of game images, solving these environments requires the algorithm to comprehend image sequences, introducing complexities in dynamics, visual features, and time. To accomplish this, our approach involved three key steps:

1. Creating a DQN solving breakout with CNN and traditional backpropagation
2. Creating a Forward-Forward model that would use Convolutional Neural Network (CNN)
3. Adapting the Forward-Forward model to the DQN

4.1 Standard DQN on Breakout

Breakout (figure 10, one of the Atari games [4], stands as great environment to test our model. It is a 2D game that involves a paddle, a ball, and a wall of bricks. The objective of the game is to break all the bricks on the screen using the ball, which bounces off the paddle. The next phase of this project involves the creation of a DQN capable of solving Breakout using standard DQN and backpropagation.

Figure 10: Breakout Atari Game [2]

4.1.1 Literature review

Atari games have been solved using DRL method for a decade now. In 2013 the DeepMind team[5], introduced the concept of using convolutional neural networks (CNNs) to approximate Q-values in Atari games. Latter on in 2015 [6], they achieved remarkable results, surpassing the performance of humans on multiple Atari 2600 games. The authors proposed experience replay and target networks as key components to stabilize training and improve convergence. The DQN has had a significant evolution and development over the past decade. While it was once an effective algorithm in the field of reinforcement learning, the landscape of RL has advanced considerably. Today, DQN is not employed in the same way as it was a decade ago as more performing algorithm have been introduced.

4.1.2 Model

Our network used has the same properties as the one from DeepMind [6]. It comprises a stack of four consecutive grayscale frames from the game as input (1x84x84). It contains three convolutional layers with 32, 64, and 64 filters, each employing ReLU activation functions and specific kernel sizes and strides. Following the convolutional layers, the network has two fully connected layers with 512 units and ReLU activation in the first, and an output layer with linear units corresponding to the possible actions in the Atari game.

Wrapper inspired from DeepMind Our custom wrapper was inspired by the one developed by DeepMind [6]. For instance, it preprocesses the image, uses 4 frame stacking, etc. All these methods are used to reduce the complexity of the environment for the network so it can learn quicker. Moreover, we added a feature which removes the 'FIRE' action and plays ten to thirty times 'No-Op' before stating a new episode. In the original game, the agent has 5 lives. However, for learning purposes, we decided to define each loss of a life as the end of an episode. This is supposed to fasten the learning process.

Replay memory and contribution In the context of image based environments like Breakout, having efficient replay memory management is crucial for optimizing memory usage. Consider the example of the DeepMind implementation [6], where one million states (or four million images) are stored in the replay memory. Such enormous data size can't be stored in the RAM without some well thought memory management. To address this challenge, a specialized approach has been implemented, incorporating smart image storage and dimensional reduction techniques. Smart image storage eliminates redundancy in image storage, ensuring that each image is stored in a memory-efficient manner without duplication. Additionally, we used the same idea as [6] for the preprocessing of the image, involving cropping, resizing, grey scaling etc. Moreover, we added, to the DeepMind preprocessing step, a new way to store the image by converting them into Boolean (Annex :19 18). This enabled a reduction of memory resources by a factor of 8 while retaining essential information. Allowing us to easily store one million frames and to train it on a standard laptop.

Encountered Issue Solving complex environments with DQN poses a considerable challenge, where slight adjustments of hyperparameters can significantly influence the learning process. To help reproducibility, we include in the annex the hyperparameters (Annex 5) we employed to achieve good results. Additionally, having an effective environment wrapper is crucial for efficient learning.

4.1.3 Results

The network manages to learn interesting strategies. During training, total reward over 300 have been achieved, enabling the network to understand advanced strategies like breaking a part of the wall in order to make the ball bounce on the other side (See Annex 20). This proves that the network is learning a good strategy. The results presented here [11] have been produced by testing the trained DQN over 100 episodes. One thing to notice is that we implemented a wrapper that marks the end of an episode whenever the agent loses a life. However, the game provides a total of five lives. In principle, our score should be the 'total reward' multiplied by five.

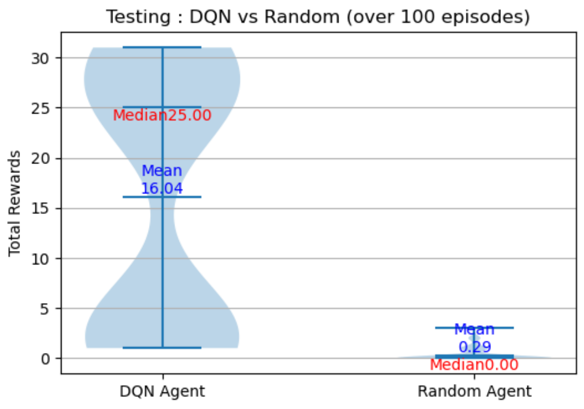


Figure 11: DQN trained with BP

4.2 CNN trained with Forward-Forward

Given that Breakout utilizes images as observations, a basic multi-layer perceptron (MLP) is insufficient for solving tasks involving image data. To tackle the complexity of environments like Breakout, employing a Convolutional Neural Network (CNN) becomes almost mandatory.

Implementing the Forward-Forward (FF) algorithm on a CNN architecture presented a unique challenge, as there were no existing public implementations or references beyond G. Hinton's paper. This part of the project relied on my interpretation of the paper, as well as insights drawn from various GitHub repositories, which may introduce potential errors.

4.2.1 Model

The network follows the same properties as the one used by G. Hinton. It consists of 3 hidden layers. "The first hidden layer used a 4x4 grid of locations with a stride of 6, a receptive field of 10x10 pixels and 128 channels at each location. The second hidden layer used a 3x3 grid with 220 channels at each grid point. The receptive field was all the channels in a square of 4 adjacent grid points in the layer below. The third hidden layer used a 2x2 grid with 512 channels and, again, the receptive field was all the channels in a square of 4 adjacent grid points in the layer below. This architecture has approximately 2000 hidden units per layer" [1].

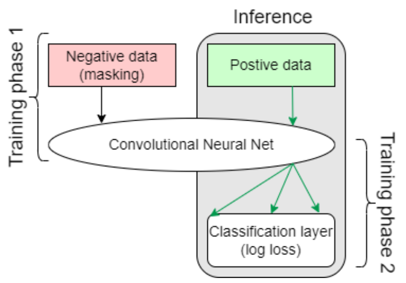
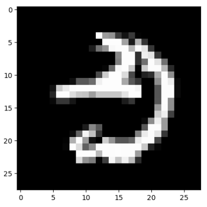
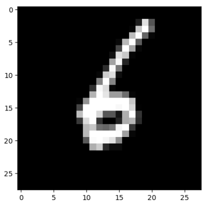


Figure 12: CNN trained with FF [1]

The training procedure consists of two distinct stages. Initially, the feature extractor is trained using FF. Following this, the classification layer is trained on top of that. After the successful training of both modules, the system is tested by sequentially passing the positive data (actual data) through these two modules. Hybrid images are used as negative data.



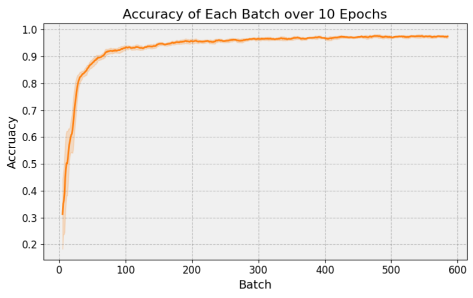
(a) Hybrid image (negative data)



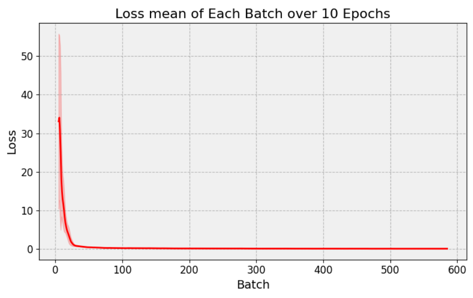
(b) Real image (positive data)

4.2.2 Results and Discussion

The model was tested using the MNIST dataset [7], a widely recognized standard for benchmarking, typically used for assessing straightforward algorithms. The graphs below shows the learning process of the classification layer once the feature extractor has been trained.



(a) MNIST FF Accruacy evolution



(b) MNIST FF Loss evolution

The results were very promising on the testing dataset with an accuracy of 97.5%. This high accuracy suggests that the feature extractor effectively identifies relevant features. Overall, the model demonstrates robust performance, closely approximating the results achieved by G. Hinton.

4.3 Forward-Forward DQN on Breakout

The algorithm designed to handle the breakout environment is also a Deep Q-Network (DQN) trained with Forward-Forward. The only distinction from the previous model [0] is the substitution of the Multilayer Perceptron (MLP) with a Convolutional Neural Network (CNN).

4.3.1 Model

The model remains the same as the one already developed for the Forward-Forward DQN on Cartpole [3.2]. The training takes place in two phases, the first one is to train the feature extractor with Forward-Forward using states from the replay memory. The second phase is to train the regression layer using standard Deep-Q Learning. The main difference from the previous model is that we are using Convolutional Neural Network instead of an MLP. This CNN comprises three convolutional layers. The first layer, has an input of 4 channels, output of 128 channels, a kernel size of 10, and a stride of 6. Subsequently, the second layer has an input of 128 channels, output of 256 channels, and a kernel size of 3. The third layer, accepting 256 input channels and producing 512 output channels with a kernel size of 2. These convolutional layers constitute the feature extraction backbone. Finally the last regression layer is just takes as input each layers output, and return the Q-values.

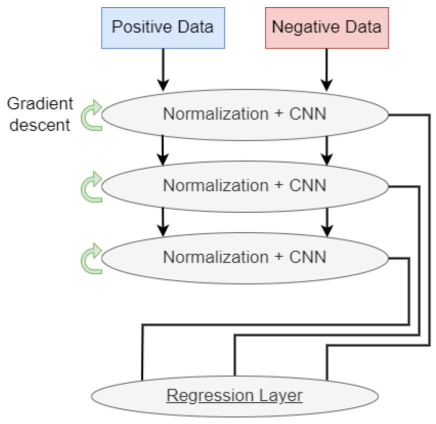


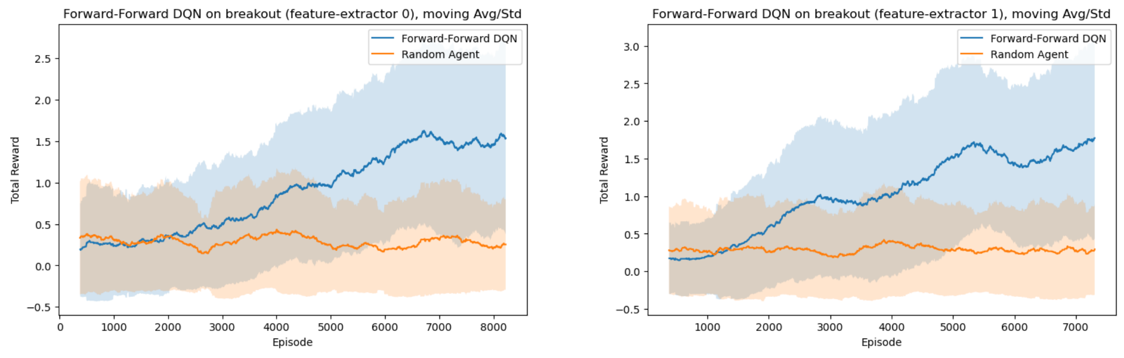
Figure 15: FF CNN used as a DQN

Negative data We used the negative data generator named 'time shuffling'. We believe the latter is supposed to be the best as it should highlight the dynamic nature of the environment. However we also thought of other types of negative data like 'Masking' or 'Blurring' that could work.

1. Time shuffling : Each states is represented by 4 observations (frames) sorted in a chronological sequence. The idea would be to change this chronological order to define negative data.
2. Masking : Using filters to mask randomly the image (like the original paper).
3. Blurring : Using a simple blurring effect, defining the negative data as being the burred one. This could potentially highlight the feature like sharp angle.

4.3.2 Early Results

The first results show us that the network is indeed learning something, which is promising given that we didn't have enough time for tuning the model. The graph (Figure 16a 16b) shows that our agent performs better than a random agent and can pretty consistently bounce the ball at least once per episode. Also the model used for this part has fewer hidden units than the one used with backpropgagtion.



(a) Feature extractor 0

(b) Feature extractor 1

Figure 16: Forward-Forward Breakout

5 Conclusion

This project serves as an exploration into the applicability of the Forward-Forward algorithm within the field of reinforcement learning. Our findings provide evidence that this algorithm can be effectively employed in model-free reinforcement learning. It has shown success in solving simple environments such as CartPole. In more complex environments like Breakout, the early results showed better performance than a random agent. Proving that the network is indeed learning something.

The objective of this project was to cover a wide range of reinforcement learning with the Forward-Forward. First, we tested it with a vanilla algorithm (see Section 2) which wasn't learning the Q-values. Subsequently, we introduced a methodology for training a Deep Q Network (DQN) with Forward-Forward (see Section 3), achieving favorable outcomes. Encouraged by these results, our focus shifted to the Breakout environment, aiming to develop an algorithm able to handle image states through the integration of Convolutional Neural Networks (CNN) and Forward-Forward. These diverse algorithms offer solutions for handling both vector and image states, which address a wide variety of environments in reinforcement learning.

As anticipated, the Forward-Forward algorithm did not surpass the performance of backpropagation in reinforcement learning. Nonetheless, exploring alternatives to backpropagation remains a valuable topic, particularly due to the lack of energy efficiency of backpropagation..

Disclosure Statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

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

[3] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013. doi: 10.1613/jair.3912.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[7] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

Annex

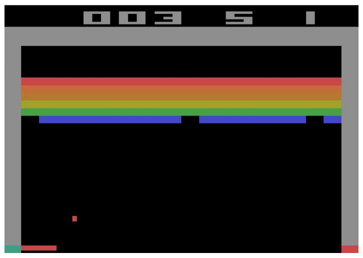


Figure 17: Raw observation

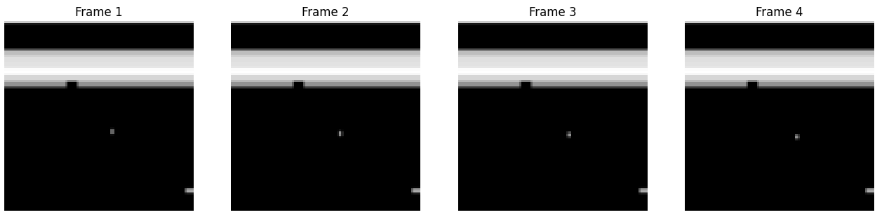


Figure 18: Without Boolean process

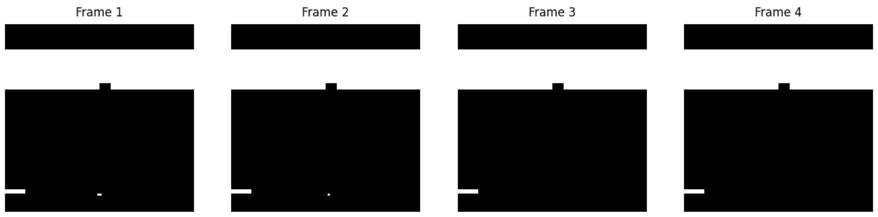


Figure 19: With Boolean process

Model Hyperparameters breakout

- Replay Memory: The replay memory had a maximum size of 300,000, facilitating efficient storage and sampling of experiences.
- Batch Size: A batch size of 32 was employed during training, enabling the model to update based on mini-batches of experiences from the replay memory.
- Discount Factor (Gamma): The discount factor, was set to 0.99, determining the weight of future rewards in the bellmann equation.
- Target Update Frequency: The target Q-network parameters were updated every 5,000 steps to ensure a stable and effective learning process.
- Exploration-Exploitation Trade-off: The exploration parameter (epsilon-greedy) started at 0.5 and underwent a linear decay over 300,000 steps, reaching a minimum value of 0.02.

- Training Procedure To initiate the learning process, the agent explored the environment for an initial 100,000 steps without updating its Q-network, allowing for diverse experience collection.
- The Optimizer used it the Adam optimizer with a learning rate of 0.0001 and epsilon value of 1.5e-4.
- The Loss function utilized during training was the Smooth L1 Loss,

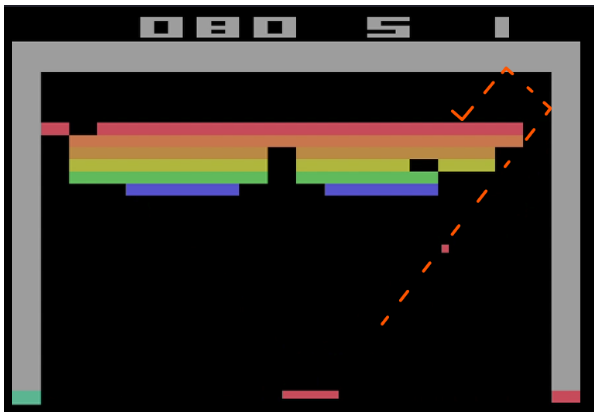


Figure 20: Advanced strategy learnt with backpropagation DQN