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

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

January 4, 2024

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 algorithm was invented. This algorithm compute 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 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 response to positive data and minimize it for negative data. This can be measured in various ways, such as the sum of squared activities. This new algorithm has shown good results for classification task but has never been implemented in 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 his simplicity and popularity as it is often used as a benchmark to compare reinforcement learning (RL) algorithm. 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 being that the forward forward has never been used to solve a regression task which is essential for most RL algorithm especially in DQL. In this report three different 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 using convolutional neural network (CNN)

The first two algorithm were implemented for the Cartpole-v1 environment and the third one, on the Breakout-NoFrameskip-v4 which is a much more complex one.

2 The Forward-Forward algorithm, an alternative to back propagation

This idea has been introduced first in 2022 by g. Hinton [1]. The main motivation for such an algorithm come 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 onto neural activities for later use in a backward pass. Moreover, the main computational cost in deep learning come from this backward propagation. Finding new algorithm to replace or improve the learning process in deep neural net, could be a revolution in the field of DL. The training cost is currently enormous for large language model (LLM). 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 theses issues and propose a new way to make a neural net learn. It works as follow "The Forward-Forward algorithm 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]. One more important thing to add is a normalization at the begging of every layers to remove any redundancy.

2.1 The Goodness

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

$$Goodness = \sum_j y_j^2$$

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

2.2 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 choose the following one due to its stability and performance:

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

Where:

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

3 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 as the state-action pair that increase the future reward, and negative data as being the state-action pair that decrease the future reward. A network that would be trained with this idea will have high goodness for action that leads to high rewards and a low goodness for action that are not going to get a lot of rewards. With this idea in mind, now the goal is to classify automatically the action that leads to high rewards. This task is not as straightforward as we can think, how can we determine good action from bad action without using traditional bellmann equation? Our first tough was the following: we run one complete episode until the agent dies and store its state action. We then determine the 5-20 states-action pairs before death as being negative data and the others action as being positive data.

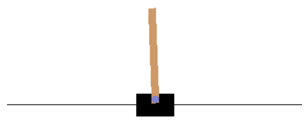


Figure 1: Cartpole [2]

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
```

3.1 Data collection for the Survival-Focused Algorithm

The FF neural network takes a state-action pair as input, the goodness is maximized for positive data and minimized for negative data. The idea is to determine positive data as the states-actions pair that maximize the survival probability and negative data as the state-actions that leads to the death of the agent.

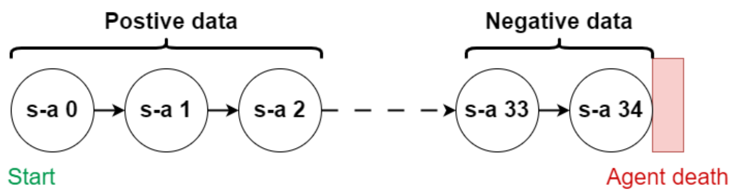


Figure 2: Positive and negative data selection

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 comprises the state-action pairs leading up to the failure state, while positive data encompasses the remaining state-action pairs.

3.2 Model

The neural network utilized in this study is a simple multilayer perceptron (MLP) consisting of three hidden layers with respective neuron counts of (10, 10, 5). 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).

This network does not produce traditional outputs; rather, it serves the purpose of assessing how it responds to specific inputs. The network's primary objective is to classify input data as either positive or negative. Strong activation of network units corresponds to positive data, while weak activation corresponds to negative data.

The inference process operates as follows: for a given state, all possible actions are evaluated. The state-action pair with the highest goodness value is selected using an epsilon-greedy exploration method.

3.3 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 episode
- Number of state-action before death defined as negative data (horizon death)
- Increasing vs decreasing horizon death
- 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 prove us that the network is indeed learning something even and that their can be room for improvement in this adaptation of the FF in reinforcement learning. This first plot proves us that the agent is performing better than random and that it is not due to luck.

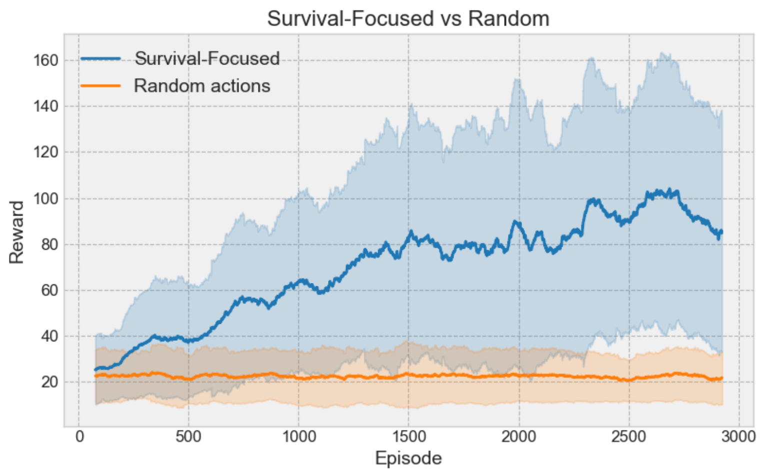


Figure 3: FF survival focused comparison with random actions

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 though 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.

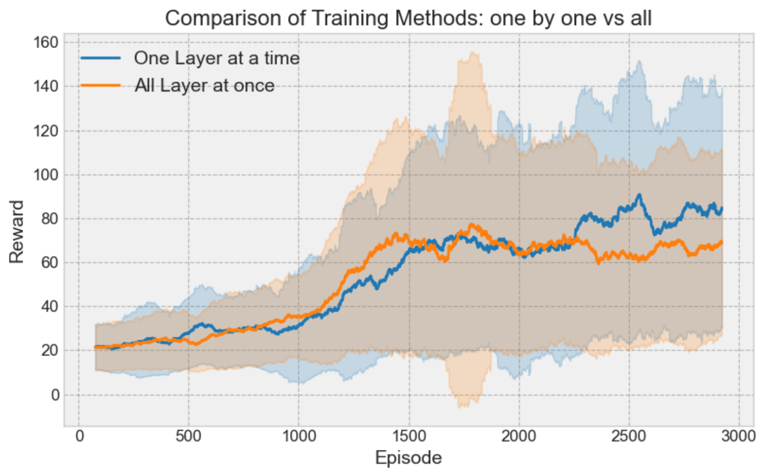


Figure 4: Training method comparison

As shown by the graph 4, the differences between these two types of training is not significant enough to conclude that one is better than the other.

Negative data Range tuning Defining negative and positive data is the most crucial part to make a network learn using the Forward-Forward. For this reason, multiple type of negative data range have been tested. For instance increasing/decresing the number of sample defined as negative over a certain ranges 2. The results shown in 5 indicates that the ranges between 5 and 20 can be utilize to have reasonable good results. Using Horizon higher than 20 will leads to inefficient learning exceptionally in constant range. Also we can notice that decreasing horizon seems to have slightly better results than increasing it.

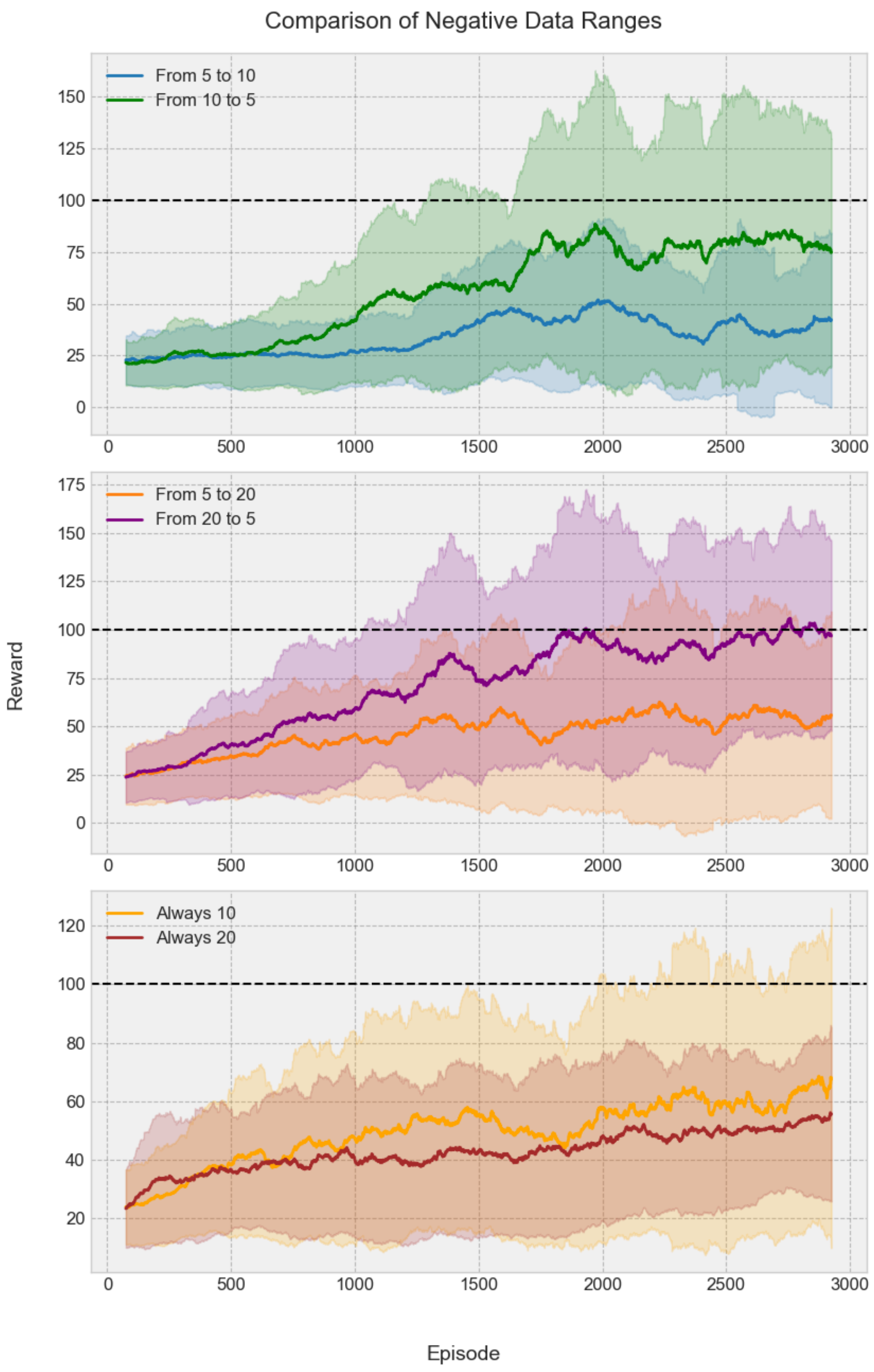


Figure 5: Horizon death range comparison

3.4 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.

4 Deep Q-Learning with forward forward

Q-learning is a crucial reinforcement learning method. The Q-value, often denoted as "Q(s, a)", represents the expected cumulative reward an agent can achieve by taking a specific action "a" in a particular state "s" and then following its optimal policy thereafter. The goal of Q-values in reinforcement learning 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. It 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)). Below is the iterative equation to estimate the Q value.

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

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 has not been done in the past. FF focused on the fact that their 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 a regression task this is really different. We don't have labels, and even if we do have target in a regression task, theses target can't be encoded directly into the input. Moreover, the FF needs a set of positive and negative data, which in the case of a regression is not well defined.

4.1 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.

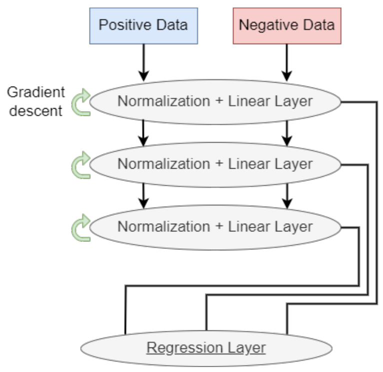

Positive Data
Negative Data
Gradient descent
Normalization + Linear Layer
Normalization + Linear Layer
Normalization + Linear Layer
Regression Layer

Figure 6: Scheme DQN with Forward-Forward

4.1.1 The creation of negative data

Creating negative data for the Forward-Forward method to address regression tasks poses a significant challenge. There is almost no available articles on this topic, making the process more reliant on intuition than mathematical proofs. Geoffrey Hinton proposed an approach for generating negative data in the context of images, involving filters that maintain micro-level coherence while introducing significant macro-level differences from real data.

In our implementation, we aimed 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 the negative data generation.

Data Shuffling Our idea refereed as ”data shuffling”, is to mix randomly the data for every dimensions. For example, to generate a new negative sample, it will selects a random value from dimension 1 of the entire dataset, followed by a random value from dimension 2, and so forth. This kind of shuffling will remove any correlation between the different dimensions which can be interesting as the Forward-Forward (FF) can learn to differentiate between data with and without such correlations. Below is an illustration of this particular procedure.

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

4.1.2 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.

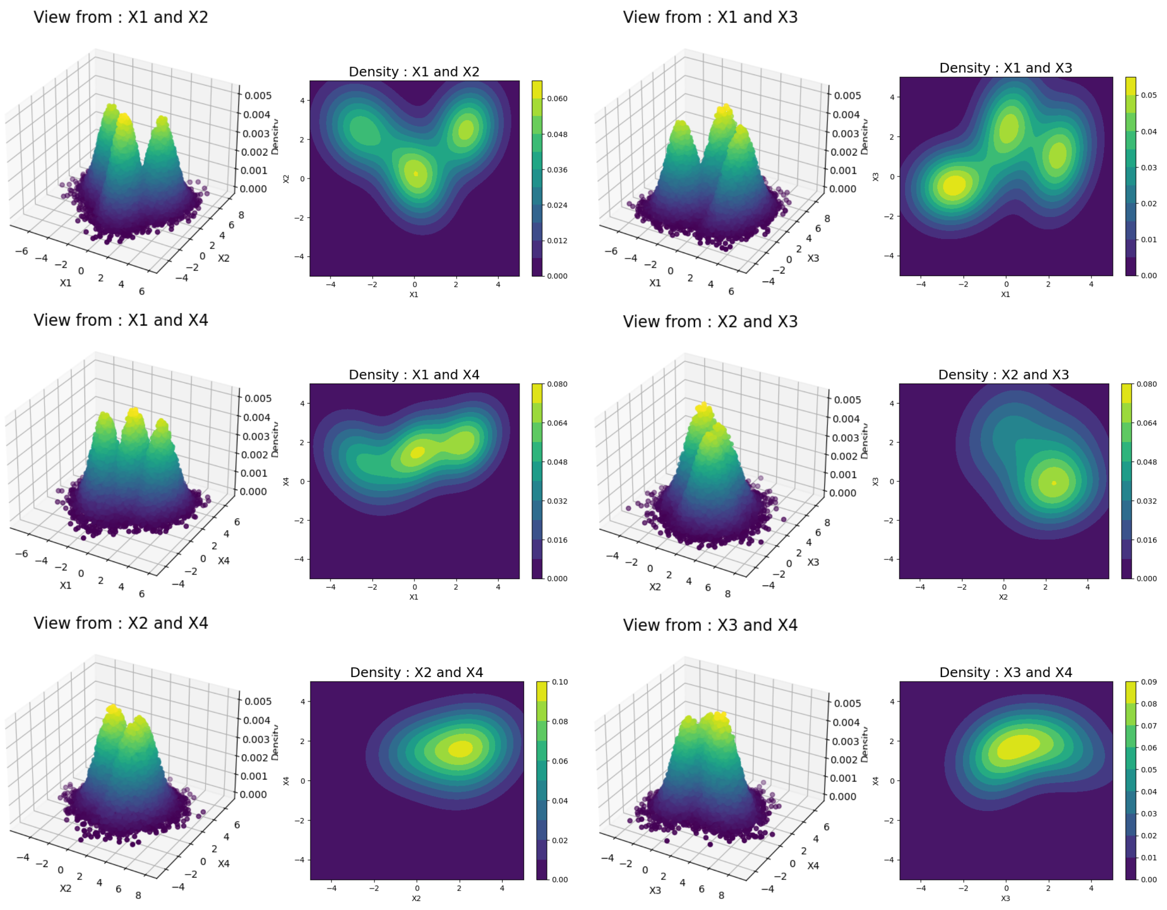


Figure 7: Dataset viewed from different dimensions

4.1.3 Model

The network here used is made of 2 parts a feature extractor and a regression layer (linear layer). The feature extractor is made with an MLP with 2 hidden layers with the following size : [input size (= 4), 10, 5]. For the regression part we used a single linear layer that takes as input the output of the feature extractor and that outputs all the Q values for a given state.

Loss function 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.

To overcome this issue, the softplus loss function was adopted. It's important to note that the softplus function from a library (e.g., torch) should be used, as it prevents potential overflow issues compared to using the exponential function.

Here's an algorithmic representation of the softplus loss:

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
```

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

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

4.1.4 Results, FF regression on custom dataset

Based on these outcomes, we notice that Forward-Forward (FF) method is almost better than 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.

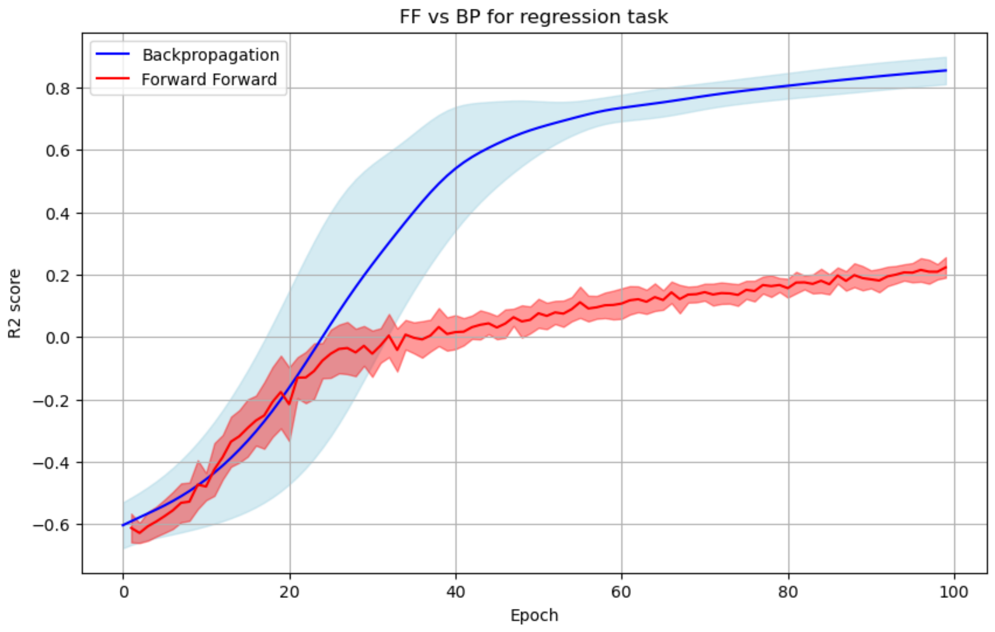


Figure 8: Performance comparison over 20 full training with the R2 score

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.

4.2 Forward Forward Q-learning

Now that we have identified a methodology that enables a neural network to learn a regression task using the Forward-Forward, the next step is to apply this approach to train the Deep Q-Learning (DQL) algorithm with FF. We employ a similar concept to train the neural network, where we begin by playing a series of random episodes to generate a dataset. These randomly generated data samples are then defined as positive data.

To create negative data, we employ the shuffling technique, which effectively eliminates correlations between dimensions within the dataset. By utilizing both positive and negative samples, our goal is to enable the forward-forward network to learn valuable features. The training of the feature extractor takes place before the DQN network is deployed. This feature extractor is trained using the states from the previous training phase.

In the final regression layer, the network outputs Q-values for a given set of states, taking the extracted features as inputs.

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 DQN using mini-batch (training step)
13:         Occasionally update target network weights
14: End
```

4.2.1 Last Regression Layer and OLS Limitations

Even if the last layer consist 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.

4.2.2 Results and Discussion

The outcomes achieved with this new approach show significantly more promising compared to the previous survival-focused method. Notably, the new method successfully solved Cartpole on certain episodes (within a 500-episode limit), which was previously unattainable with the survival-focused method. However, when we compare this new method to the standard DQN with backpropagation, it becomes apparent that the learning process is less stable. These results are indeed really promising and shows significant improvement compare to the Survival-Focused method. The only issue come from the fact that the learning process is less predictable than a traditional DQN with backpropagation.

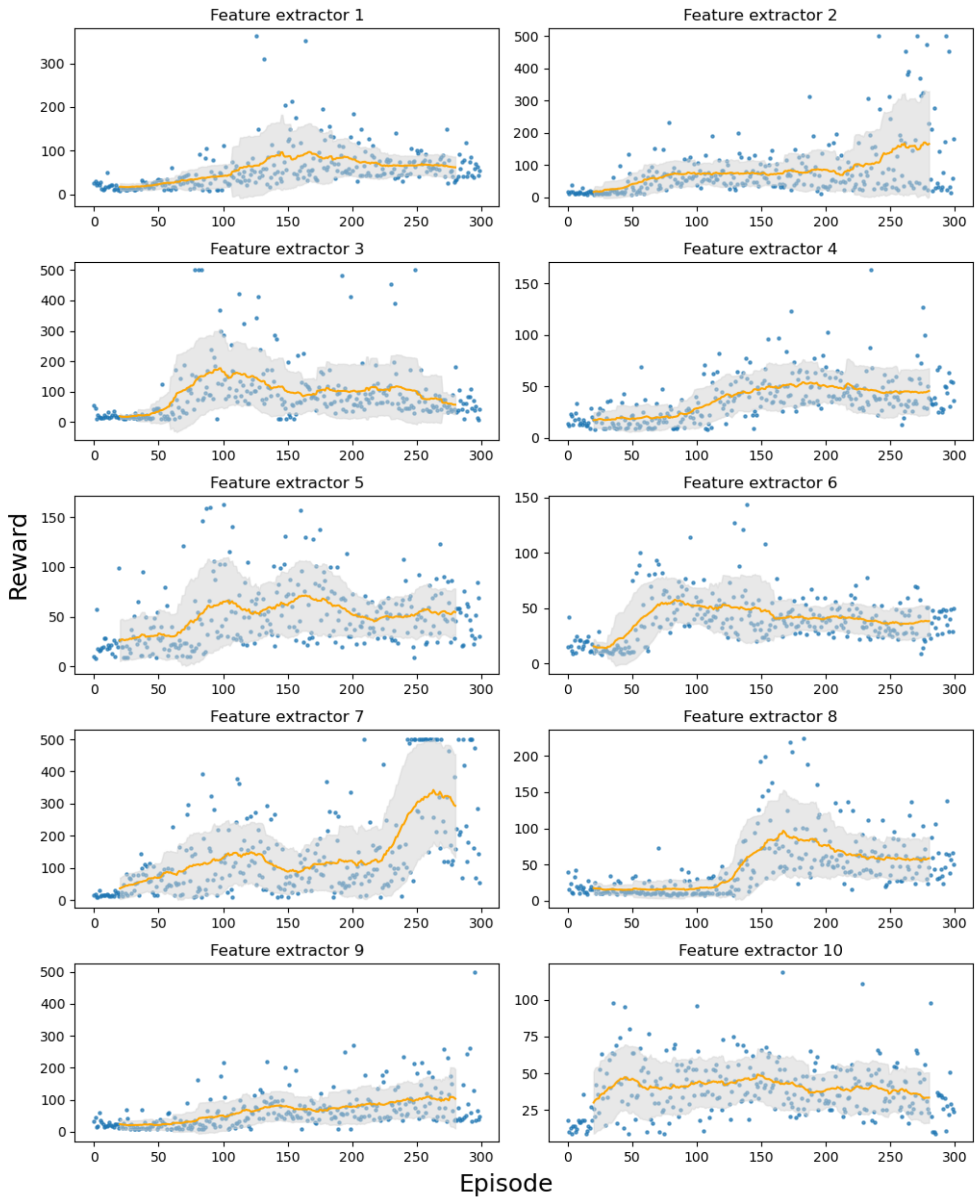


Figure 9: Reward Evolution of the DQN trained with FF

The feature extractor performances were very hard to predict.In certain cases, it didn't led 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.

5 Forward-Forward in complex environement

The current objective is to extend the FF learning algorithm to more complex environment and see if some interesting results can be achieved. Moreover, evaluating our methods on Atari games serves as a benchmark to prove the robustness of the algorithms. The data that Atari games provides is solely images of the game. For the algorithm to solve such environment, it has to understand the sequence of images that are provided. This add diverse complexity such as dynamics, and visual features and time, making these environment a real challenge for DRL. This section will be divided in tree main topics:

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

5.1 Standard DQN to solve Breakout

The Atari games [4] are a set of simple 2D video games that are widely used in the field of reinforcement learning. They provide simple environment with images as inputs. To solve such environments using reinforcement learning, Convolutional neural network (CNN) are used. The next step of this project is to adapt the FF to an algorithm capable of solving breakout game. To be able to do so, the first step will be to find a way of making CNN trainable with forward forward which hasn't been done in the past.

Figure 10: Breakout Atari Game [2]

5.1.1 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.

Wrapper inspired from deepmind The custom wrapper done in this paper was inspired by the one developed by deepmind [6]. It removes the 'FIRE' action, plays ten to thirty time 'No-Op' before stating a new episode, uses 4 frame stacking etc. All these method are used to reduce the complexity of the environment for the network so it can learn quicker.

Replay memory and contribution In the context of complex environments like Breakout, efficient replay memory management is crucial for optimizing memory usage. 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, dimensional reduction involves preprocessing images and converting them into Boolean representations 15 14, reduce significantly the memory resources while keeping essential information.

Issue encountered Unfortunately, this part on breakout was much more challenging than expected. The code is available on GitHub, I think the problem comes from some hyper-parameter tuning as it is often the case in reinforcement learning.

5.2 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.

Previously the implementation was done with on a MLP, to deal with images we should use Convolutional Neural Network (CNN). 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.

5.2.1 Model

The model used follows the same properties as the one cited in the paper. 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].

The training procedure consists of two distinct stages. Initially, the feature extractor is trained using FF. Following this, the already trained feature extractor is employed in the training of the classification layer. After the successful training of both modules, the system is tested by sequentially passing the positive data (actual data) through these two modules.

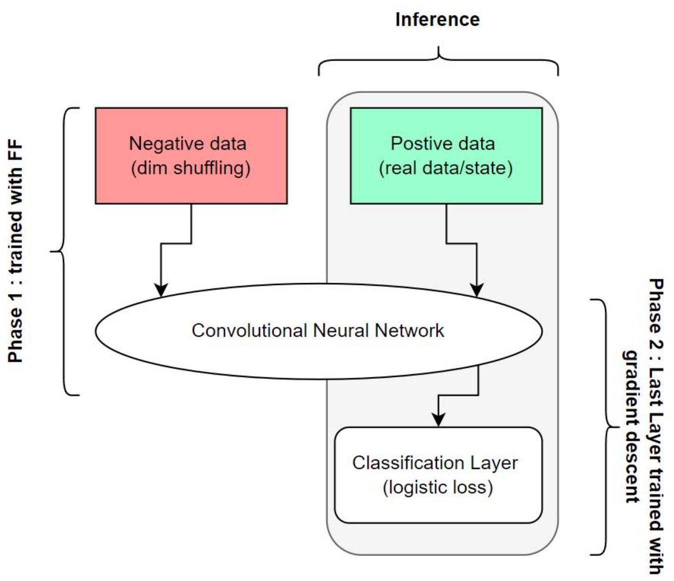
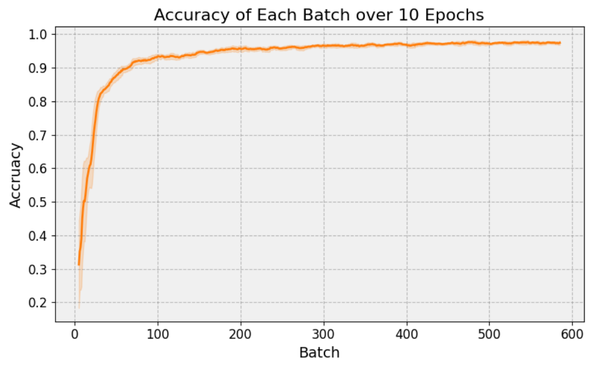

Inference
Negative data (dim shuffling)
Postive data (real data/state)
Convolutional Neural Network
Classification Layer (logistic loss)
Phase 1 : trained with FF
Phase 2 : Last Layer trained with gradient descent

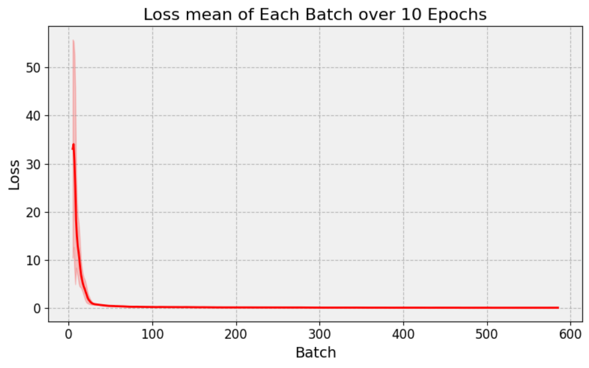
Figure 11: CNN trained with FF [1]

5.2.2 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

We also get great results with a testing accuracy of 97.5% on the MNIST dataset. This high accuracy is also a positive indicator, suggesting that the feature extractor effectively identifies relevant features. Overall, the model demonstrates robust performance, closely approximating the results achieved by G. Hinton.

5.3 Advanced Deep Q-Network

This part has not been completed and would be subject to future work. The following section will propose a way to generate negative data for such complex environment but we didn't had time to test it.

5.3.1 Negative data

As already mentioned in the previous chapter, the generation of negative sample is one of the most crucial part in the FF. In the case of breakout, we came with three different idea for it:

- 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.
- Masking : Using filters to mask randomly the image (like the original paper).
- Blurring : Using a simple blurring effect, defining the negative data as being the burred one. This could potentially highlight the feature like sharp angle.

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

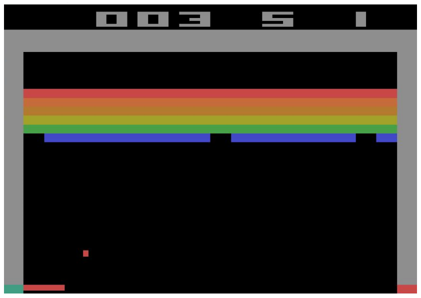


Figure 13: Raw observation

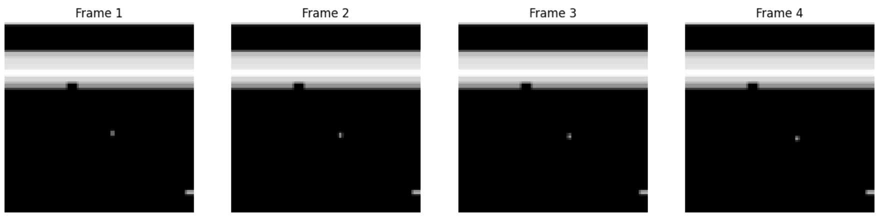


Figure 14: Without Boolean process

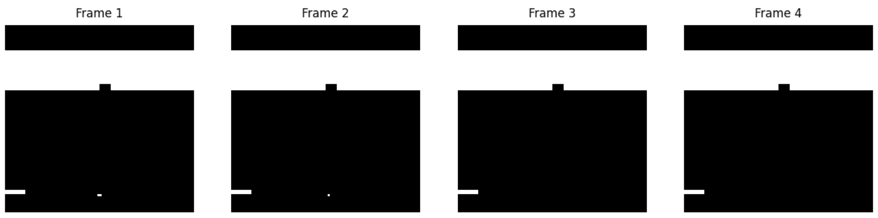


Figure 15: With Boolean process