École Polytechnique Fédérale de Lausanne

Forward-Forward algorithm evaluation in Deep Reinforcement Learning

by Léon Delachaux

Semester Project Report

Approved by the Examining Committee:

Prof. Roberto Castello
Project Advisor

Giulio Romanelli
Project Supervisors

October 20, 2024

Abstract

Deep reinforcement learning is a powerful approach utilized in various domains, aiming to tackle challenging tasks by training neural networks to interact with environments. These environments range from strategic games like chess and go to more practical applications such as natural language processing. In a recent publication by Geoffrey Hinton, a novel training method called Forward-Forward was introduced as an alternative to the conventional backpropagation algorithm. This project aims to investigate the suitability of Forward-Forward training for neural networks within the context of deep reinforcement learning. In this report, I am describing the experiments that were performed to try to solve the environments of CartPole and Breakout by training an agent with the Forward-Forward algorithm.

Contents

Chapter 1

Introduction

Deep reinforcement learning is a fusion of reinforcement learning (RL) and deep learning. RL tackles the challenge of enabling computational agents to learn through trial and error in decision-making processes. By incorporating deep learning, it is possible to empower agents to make decisions based on unstructured input data, eliminating the need for manual construction of the state space. One key advantage of deep RL algorithms is their capability to process extensive inputs, such as every pixel displayed in a video game, and determine optimal actions to achieve a specified objective, such as maximizing the game score.

The Forward-Forward (FF) algorithm has been introduced in 2022 in a paper from Geoffrey Hinton. The paper discusses the limitations of the backpropagation algorithm, which is the common way for training neural networks, and explores this alternative algorithm.

The paper argues that backpropagation is implausible as a model of how the brain learns and that it requires perfect knowledge of the forward pass computation. In contrast, FF can be used even when the precise details of the forward computation are unknown, can learn without storing the neural activities or stopping to propagate error derivatives, and may be superior to backpropagation as a model of learning in the cortex and also when making use of very low-power analog hardware. However, the paper adds that Forward-Forward is somewhat slower than backpropagation and does not generalize as well on some toy problems.

The paper presents the Forward-Forward algorithm by training a neural network with the MNIST dataset to classify digits. It shows two examples, one that uses supervised learning and another that uses unsupervised learning.

In this project, I have conducted experiments on two simple environments: CartPole and Breakout. In these experiments, I am trying to train a deep learning agent with FF. I extracted results during training and I am describing the experiments and the results in this report.

Code is available at `https://github.com/leondelachaux/DRL_project`.

Chapter 2

Background

Reinforcement learning

Reinforcement learning is a machine learning approach to enable agents to learn optimal decision-making strategies in dynamic environments.

RL relies on the Markov Decision Process (MDP), which models sequential decision problems. At each time step, the environment is in some state s, and the decision maker (agent) may choose any action a that is available in state s. The environment responds at the next time step by moving into a new state s', and giving the agent a reward r. For example, if the environment is a game of chess, then the reward could be 1 when the agent wins by checkmate and -1 when it loses.

In RL, we train an agent to find the best policy to maximize cumulative rewards. The policy π is a mapping from the state of the environment to a probability distribution of the possible actions and the cumulative reward is the sum of the rewards that are accumulated over time by following the actions generated by the policy, starting from an initial state.

Reinforcement learning starts with a prediction problem. We can define the value function, which is the expectation of the discounted cumulative reward starting from the state s following policy π.

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right]$$

If we are playing an episode (ex: playing a game of chess), t is the number of steps that we have done, so that s_t is the current state of the environment and $\{r_{t+k+1}\}_{k=0}^{\infty}$ are the future rewards after k new steps. γ is the discount factor, which is a parameter between 0 and 1 that is used to

balance how willing the agent should be to accept a small reward in the short term to achieve a bigger one in the long run. The discount factor can also be useful if the game does not always end, so usually, $\gamma = 0.99$ so that we are sure the above sum is finite.

In the prediction problem, this value function can be determined by solving the Bellman equation:

$$V_\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma V_\pi(s_{t+1}) \middle| s_t = s \right]$$

The Bellman equation plays a central role in RL, because RL algorithms improve the agent's policy by maximizing this value function.

The same way that we define the value function, we define the Q-function.

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right]$$

The Bellman equation can also be written by replacing the value function with the Q-function, and Q-learning, which is introduced below, is a RL algorithm that improves the agent's policy by maximizing this Q-function.

There are several ways to perform deep reinforcement learning. For example, the algorithms can be deep Q-learning, A3C, A2C, DDPG or PPO. Today, there are libraries to perform deep reinforcement learning. In particular, Stable Baselines3 can be very useful to train deep RL agents, this library implements all the deep RL algorithms and all that is needed is the environment, then the methods from SB3 can deal with the training.

In this project, we decided to start by implementing the deep Q-learning algorithm. Deep Q-learning consists in training a neural network to estimate the Q-function, the neural network is called deep Q-network (DQN). The figure below illustrates deep Q-learning with two DQNs, the policy net that will be optimized after each time step and the target net which slowly track the policy net as a soft update is applied on its weights so that it is more stable.

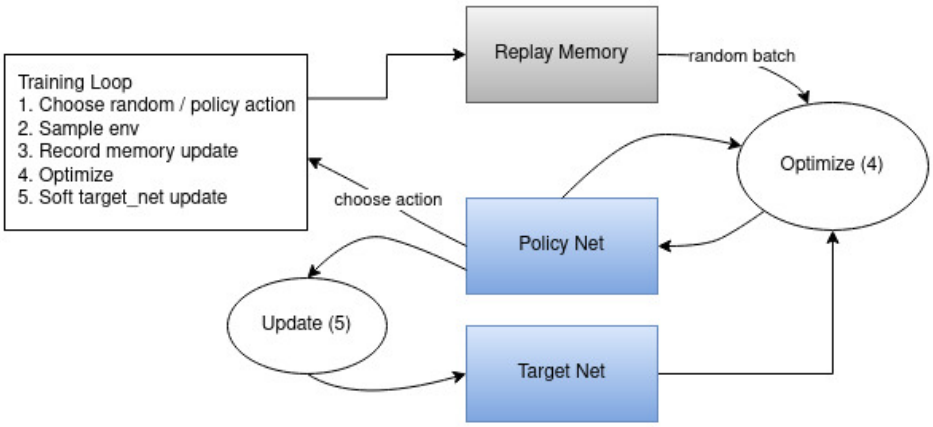

Figure 1: Deep Q-learning idea [2]

In classic Q-learning, the Q-function is updated for a state-action pair based on the immediate reward received and the estimated future reward obtained by taking the optimal action in the next state. This involves the following update rule:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

with s' being the state after choosing action a in state s.

The Q-function can be stored in a lookup table, which might be computationally expensive for large state spaces, but using a DQN to approximate it allows for more efficient computation and generalization to unseen states. Instead of applying the update rule, we will define the loss and perform backpropagation so that the neural network learns to minimize it.

$$L = \left| Q(s,a) - \left(r + \gamma \max_{a'} Q(s',a') \right) \right|$$

The DQN takes the state as input and outputs Q-values for each possible action. The agent then chooses the action with the largest Q-value. However, there are some challenges when training the DQN, such as instability or overestimation of Q-function. To address these issues, techniques such as experience replay and target network are used.

Experience replay involves storing the agent's experience (i.e. state, action, reward and next state or none in the terminal case) in a replay buffer and sampling batches of experiences randomly during training. This helps to decorrelate the data and prevent the network from overfitting to recent experiences.

Target network involves using a separate network to generate the target for updating the policy network. Like the policy net, the target net is updated at each time steps but with a soft update $\theta' \rightarrow \tau\theta' + (1-\tau)\theta$ with τ a hyperparameter, θ and θ' being the weights of the policy net and the target net.

Overall, deep Q-learning is a powerful approach for training agents to make decisions in complex environments, and has been successfully applied in a wide range of domains such as robotics, video games, or autonomous driving so it will be the baseline of this project.

Forward-Forward [1]

The first step of the Forward-Forward algorithm is to have or create positive and negative data.

To give an idea, the positive data is generally the data that we would feed the neural network if we were using backpropagation, and the negative data is generally something that the neural network should learn to "ignore".

During the training, instead of one forward pass and one backward pass like backpropagation, with FF two forward passes are performed at each epoch, hence the name Forward-Forward. One forward pass is performed with the positive data and the other is performed with the negative data.

The optimization part is the following, after the computation each layer is updated separately, the objective is the the goodness, and one way to define the goodness is to sum the activations computed by this layer.

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

Here, σ is the sigmoid function, $\{y_j\}$ are the activations and θ is a threshold that we can fine-tune.

The goal of the process is to minimize the goodness of each layer for the negative data and maximize it for positive data. For this, the layer is updated with gradient descent, using a learning rate, the difference with backpropagation is just that the error of each layer is not back-propagated.

It is important to note that the goodness of a layer does not depend on the goodness of the previous layer because the activations are normalized before being the input of the next layer.

There is not really a unique and versatile solution to have the positive and the negative data, but the paper presents two examples one for unsupervised learning and the other for supervised learning.

Unsupervised learning Forward-Forward

The paper proposes FF for unsupervised learning with the use of contrastive learning to improve the performance of a neural network in identifying differences between images. The approach involves mixing together images of different digits and training a network to disregard their similarities, thereby enhancing its ability to identify dissimilarities.

To generate negative data, the paper introduces masks that consist of large regions of ones and zeros. These masks create images with distinct long-range correlations but similar short-range correlations, providing a training signal for the network to learn to disregard shared characteristics between digits.

Indeed, the main idea is to train a network capable of effectively ignoring similarities between digits, thereby improving its ability to detect and distinguish their differences.

Additionally, the paper suggests that a single linear layer followed by a softmax function can be trained to predict the labels of the images.

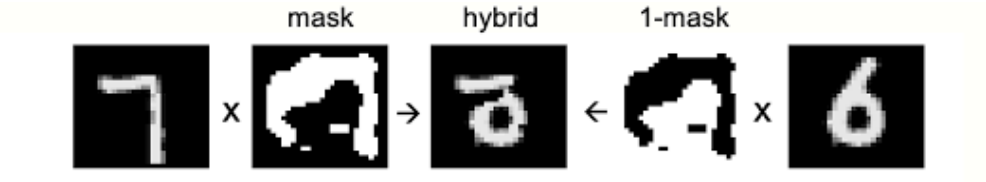

Figure 2: How negative data is created [1]

Supervised learning Forward-Forward

The main idea presented in the paper is to include label information in the input data before training. This is achieved by creating positive data, which consists of images with the correct label, and negative data, where images are assigned random labels. The label information is incorporated by replacing the initial pixels of the image with a one-hot encoded representation of the label.

During training, the network learns to assign large weights to the one-hot encoded pixels, allowing it to correlate the label information with the internal structure of the image. This helps the network to focus on features that are relevant to the given label.

During inference, there are two approaches suggested. The optimal approach involves forwarding the input with each possible label, accumulating the goodness of the network's responses, and selecting the label with the highest accumulated goodness. The suboptimal approach involves replacing the one-hot encoded label pixels with a value of 0.1, and training a linear layer followed by a softmax classifier to classify the digits.

It's important to note that although the paper focuses on image inputs, the concept can be applied to vector inputs as well. In the case of images, they are flattened before being input to the network. The example in the paper uses a simple neural network with fully-connected layers, without convolutional layers.

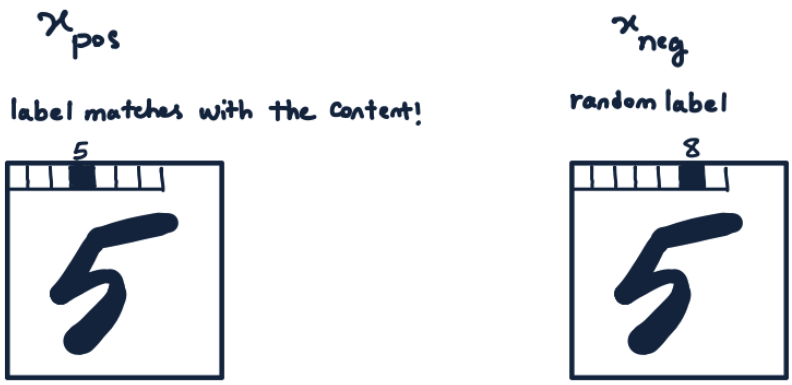

Figure 3: Positive inputs vs negative inputs [3]

The problem described above is a classification problem, but in RL we can be interested in the regression problem to estimate the future reward. The paper does not show anything about it, and it is not mentioned in the future work section.

Further investigation and future work

The paper delves into a detailed evaluation of the method and proposes avenues for improvement. It identifies two main weaknesses: the inability of later layers to influence earlier layers' learning and the potential for errors in activation normalization. The authors put forth suggestions to enhance these aspects, but further investigation is required to validate their efficacy.

Furthermore, the paper outlines additional areas for future exploration. One such idea involves leveraging a Generative Adversarial Network (GAN) to generate negative data. Additionally, the researchers plan to investigate the goodness function, activation function, and other potential enhancements to refine this technique.

Notably, the paper exhibits a focused research direction on modeling the brain. It contemplates scenarios where positive data could be fed during wakefulness and negative data during sleep, either through brain-based machine learning or when studying the impacts of sleep deprivation.

Chapter 3

Design (Framework description)

We are conducting experiments with CartPole and Breakout environments.

CartPole

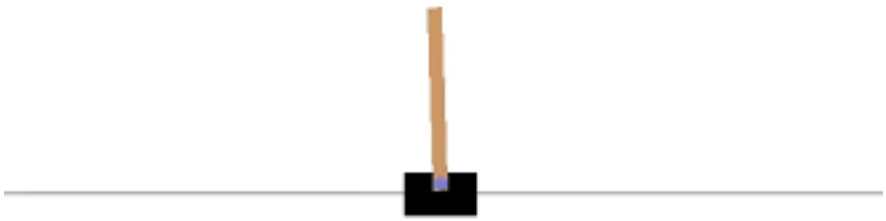

Figure 4: CartPole environment

In the CartPole environment, I conducted experiments using deep Q-learning as a baseline to establish reference results. Then, I had to find a way to use FF, and decided to classify the state according to the best possible action.

The idea behind is to include the action associated with each state as an additional input. During training, I fed the network with the last N state-actions as negative data, while the rest of the data was assigned as positive.

Normally the last action are assumed to be worst than the first. At least, they are supposed to be

leading to a loss.

After updating the network, I repeated the process with a lower epsilon value, implementing decreasing exploration:

$$\epsilon(n) = \max\left\{\epsilon_{min}, \epsilon_{max}\left(1 - \frac{n}{n^*}\right)\right\}$$

with n being the number of epochs already done, and n^* being the total number of epochs to be done for the training.

In the case of CartPole, the idea is to assume that the last actions are more likely to be losing moves. This assumption is probably not exactly true, but it can be interesting to find out if it is possible to have some improvement.

In fact, I also tried an approach in which the state-action can be assigned as negative data if the possible next-state-actions had low goodness, but this approach provided worse results, so I decided to continue with the previous idea.

For the evaluation, I collected the durations of the episodes during training and plotted them for deep Q-learning and FF to compare them. For CartPole the total reward is equivalent to $0.1 \cdot$ duration, because the environment delivers a reward of 0.1 at each time step as long as the stick is not falling.

Breakout

Figure 5: Breakout environment

In the Breakout environment, the state is larger in terms of the amount of data available, but it can be problematic if there is a lot of features that do not contain essential information. To address this issue, I preprocessed the state by cropping the image, resizing it, and converting it to black and white.

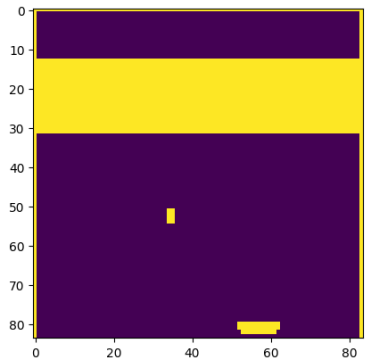

Figure 6: Breakout preprocessed

Furthermore, in Breakout the next state depends on factors like the velocity of the bullet and changes in its direction. To account for this, I stacked four frames together and passed them as input to the neural network. [4]

For the FF approach in Breakout, I used a similar method as in CartPole. I employed a neural network and passed the stacked frames, flattened them, added the action, and used this as input. I accumulated the goodness and considered the last state-action as negative data, while the initial state-action served as positive data.

For the evaluation, I collected the rewards during training for A2C and FF and produced the plots to compare.

Chapter 4

Implementation (experimental setup)

The first experiment I conducted is using a DQN to solve the CartPole problem. In the DQN experiment, the neural network consisted of three linear layers, with the second one that is 128 inputs to 128 activations. I utilized a replay memory of size 10,000, a learning rate of $2 \cdot 10^{-4}$, and a batch size of 128. The exploration-exploitation trade-off parameter, epsilon (ϵ), decreased exponentially from 0.9 to 0.05 during training. The target network update rate, tau (τ), was set to 0.005. I played a total of 600 games in this experiment.

The second experiment is to use FF to train a neural network to determine the best action in CartPole. For this experiment, the neural network had two layers, each with 100 activations. I used a learning rate of 0.0003 and trained the network for 200 epochs. Each epoch involved using data from 20 games, resulting in a total of 600 games played.

Moving on to the Breakout game, I encountered some challenges in obtaining results. Before training, I needed to preprocess the state, to limit the data to the more essential information. I cropped the image, I cropped 7 pixels on each side, 31 on top 14 at the bottom, then resized it and converted it to black and white to end up with an 84x84 image.

I attempted different implementations for Breakout. I attempted to solve it with Deep Q-Learning, and I also performed tests using Stable Baselines3 with another algorithm called Advantage Actor-Critic (A2C).

With deep Q-learning, I used a convolutional neural network. With A2C, I trained the model for 600000 timesteps, also with a convolutional neural network to determine the policy.

For Breakout with FF, I followed a similar approach as with CartPole. The neural network had two layers, each with 300 activations, and I used a learning rate of 0.00003. Note that a game of Breakout, a game consists of 5 episodes (the player has 5 lives), so we just need to play 4 games to have 20 episodes. In the end, I trained it for 200 epochs which corresponds to $20 \cdot 200 = 4000$ episodes.

Chapter 5

Evaluation

CartPole

Let's begin by examining the reward plot during training with deep Q-learning.

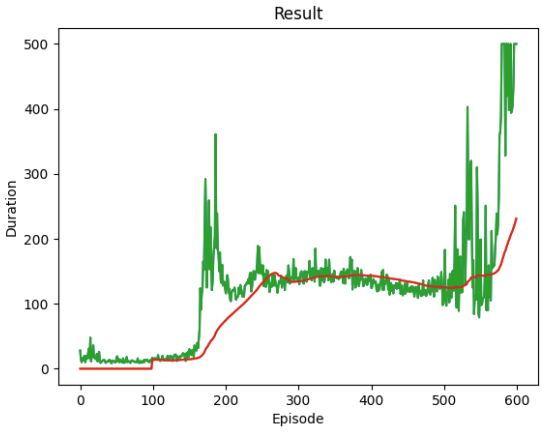

Figure 7: deep Q-learning training

The network learns quickly and eventually reaches the maximum possible reward.

Next, we have two plots that utilize the method with the Forward-Forward algorithm. The first plot represents a training session where the last 3 state-actions are used as negative data. The second plot depicts a training session where the number of last state-actions used as negative data is set to 30% of the average number of moves in the previous epoch.

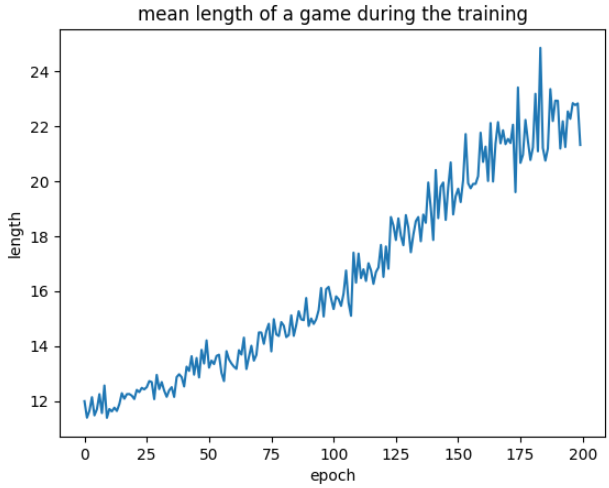

Figure 8: FF Training of a CartPole agent with N=3

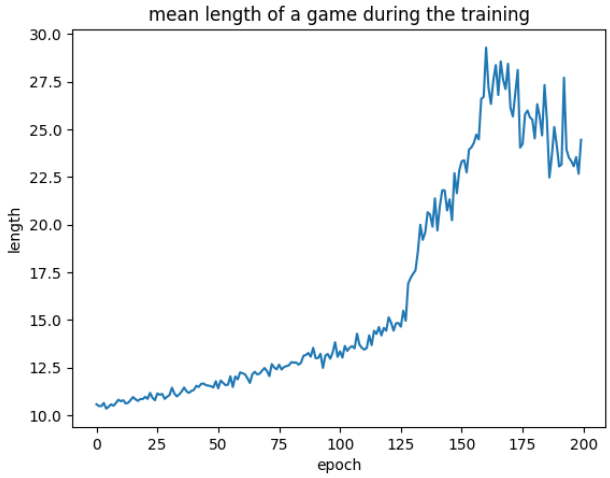

Figure 9: FF Training of a CartPole agent with N = 30% of the whole game

Although there is some improvement during training, this is not comparable with the results of a DQN. An agent with random actions achieves an average duration of 11, while the best agent we managed to train using the FF technique does not exceed 30, while the DQN quickly achieves 10 times better and even goes up to the maximum number of steps because the CartPole environment from Gymnasium can not have an episode longer than 500.

Breakout

For Breakout, the technique with FF is not expected to work more effectively. Breakout is considerably more complex to solve compared to CartPole.

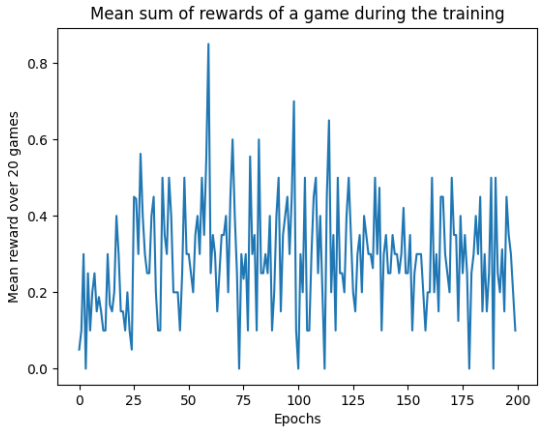

Figure 10: FF Training of a Breakout agent with N = 10

The $N = 10$ was chosen because 20 seemed to be the minimum of an episode for breakout.

We can compare with A2C and see the difference.

Figure 11: Training of a Breakout agent with normal a2c

Despite unstable results during training, the A2C agent manages to learn and to achieve large rewards, while the FF-trained agent still looks like a random agent.

Chapter 6

Conclusion

To conclude, we have seen that it was possible to train an agent with pure FF, but for now this is not enough to outperform algorithms trained with backpropagation, but it suggests that further research should be conducted on this.

It is surely possible to do supervised learning to learn the best actions, but if we really want to perform reinforcement learning we have to create an algorithm different from what is existing, or find a way to apply FF in the case of a regression problem.

If FF could be applied to regression, then the classic deep RL algorithms could be adapted to train neural networks with FF. Else, we would need to find a way to isolate the bad actions after the episodes to pass them as negative data, which is difficult to do.

In the paper, the point of FF is not only to solve complex problem, but more importantly to offer more comprehension of the human cortex. For the wider range of neural network uses, it is too early for FF to replace backpropagation.

Bibliography

[1] Geoffrey Hinton, *The Forward-Forward Algorithm: Some Preliminary Investigations*, 2022

[2] Adam Paszke, Mark Towers, *Reinforcement learning (DQN) tutorial*
`https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`

[3] Mohammad Pezeshki *pytorch_forward_forward*,
`https://github.com/mohammadpz/pytorch_forward_forward`

[4] Gabriel Andersson and Martti Yap *Learning to Play Breakout Using Deep Q-Learning Networks*
`https://rssalessio.github.io/teaching/theses/gabriel_martti_qlearning.pdf`