TIPE : L'ÉLECTROSTIMULATION AU SERVICE DU SPORT POUR TOUS

Mohamed Aymane
MOUSSADEK

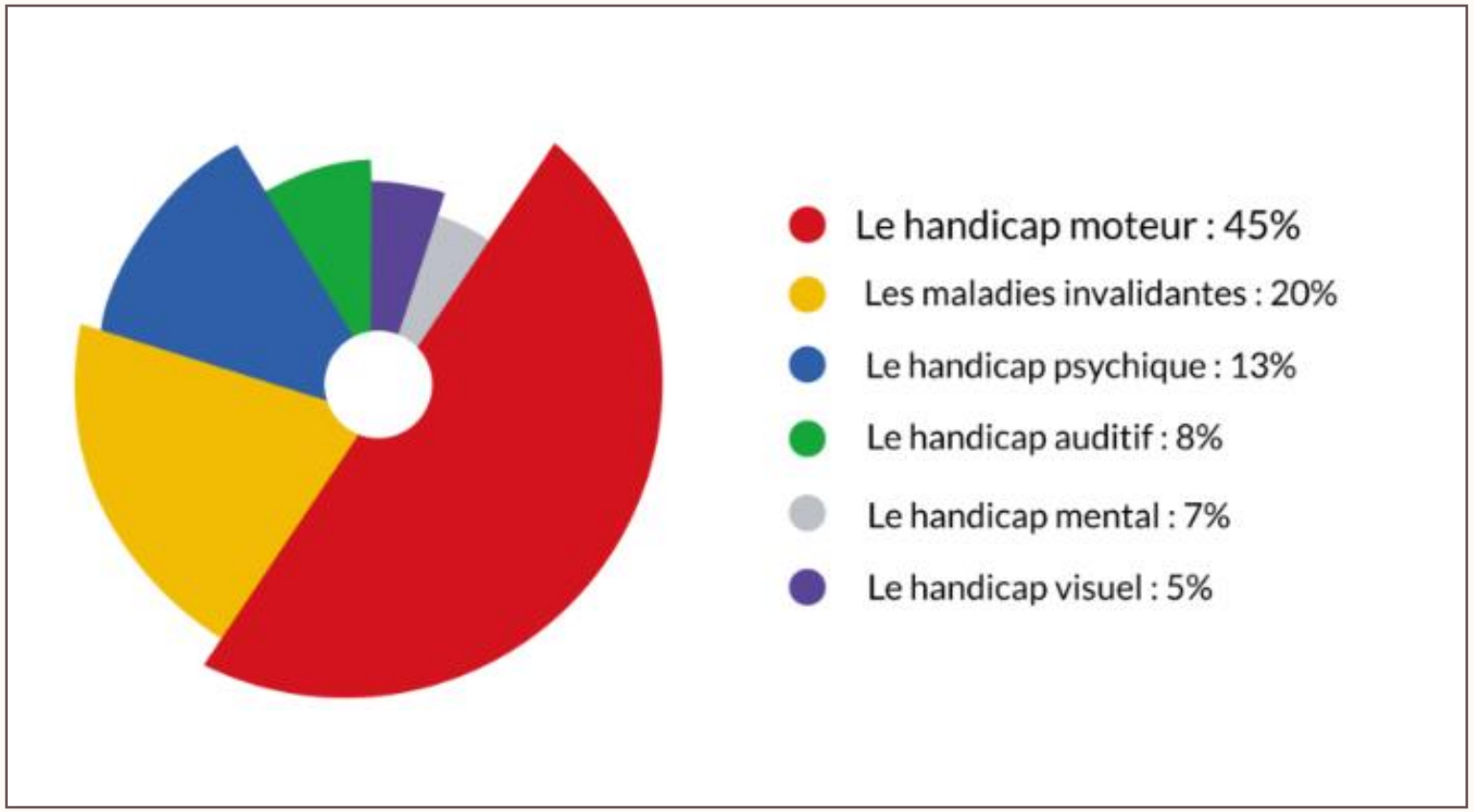
Session 2023

N°18367

SOMMAIRE

MOTIVATION ET PROBLÉMATIQUE

Rendre l'activité physique accessible à tous et améliorer les performances des sportifs confirmés a toujours été un enjeu majeur dans le monde du sport. Cependant, si de nombreux progrès ont été réalisés dans de nombreux domaines sportifs, il reste encore beaucoup à faire, notamment dans le développement des infrastructures et des moyens permettant aux personnes handicapées de pratiquer le sport.



https://numericoach.fr/zoom-sur-le-handicap-et-laccessibilite-pour-tous/

Actuellement c'est 1.3 milliard de personnes atteintes d'un handicap grave dans le monde, souvent du a un accident… et leur empêcher la pratique d'une activité sportive. Cependant la pratique d'un sport essentielle pour la sante morale, physique mais aussi pour la réinsertion sociale de ces personnes, des lors trouver des solutions à ce problème est un besoin pressant pour nos sociétés.

SOLUTION PROPOSÉE

- L'une des solutions à ces différents enjeux et qui a révolutionner le monde du sport ces dernières années et l'électrostimulation musculaire (FES). L'électrostimulation musculaire est un procédé de stimulation des fibres musculaires par l'émission d'onde électrique modulées permettant de réaliser un mouvement précis sans l'intervention du système nerveux du patient. Les domaines d'application de cette technologie sont nombreux, de l'amélioration des performances sportives d'athlètes professionnels leur permettant de méthodes d'entrainement nouvelles, à son utilisation pour permettre à des personnes atteintes d'un handicap de pouvoir faire du sport de nouveau !

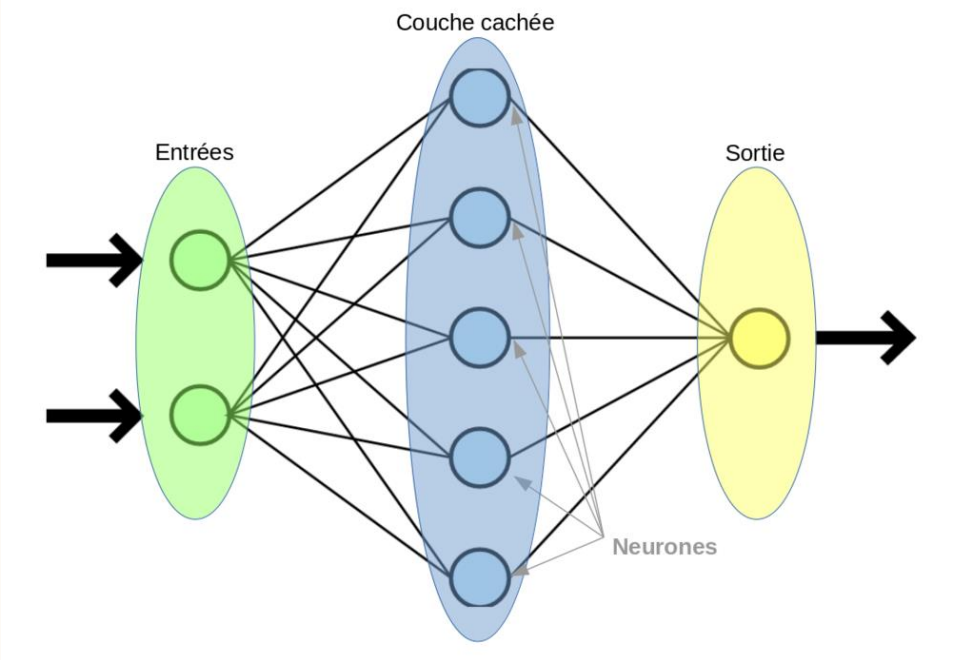
FES BIKE

Le FES bike est un tout bonnement un vélo stationnaire ou non stationnaire équipe d'un ordinateur fournissant des impulsions électriques pour un patient suivant une thérapie. Grace à un ensemble de capteurs et à un processeur qui traite les données, le système doit parvenir à émettre les bonnes impulsions au bon moment pour permettre à un patient de pédaler naturellement malgré la faiblesse, la paralysie ou d'autres problèmes fonctionnels.

Presentation title

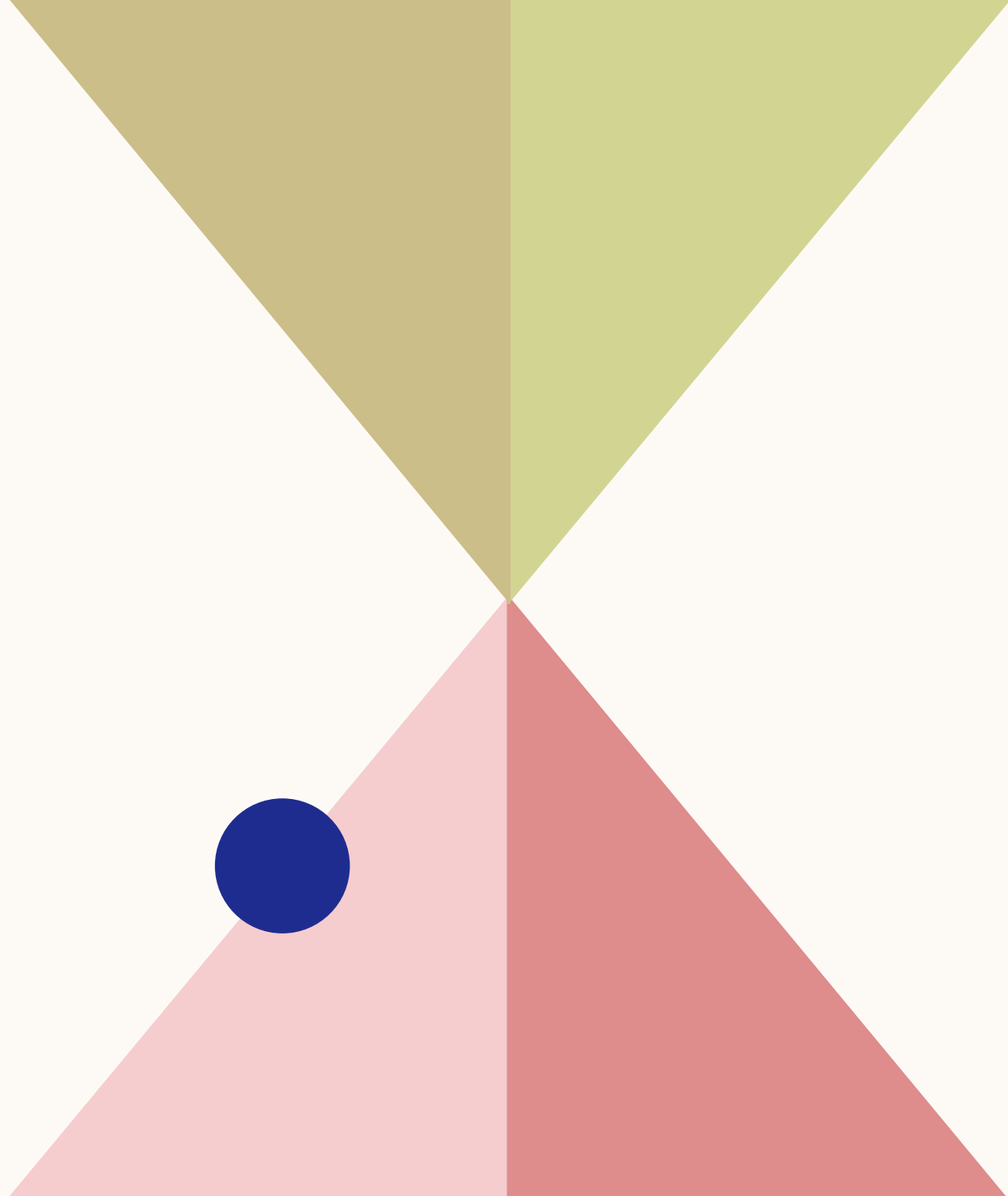
RESEAU DE NEURONES

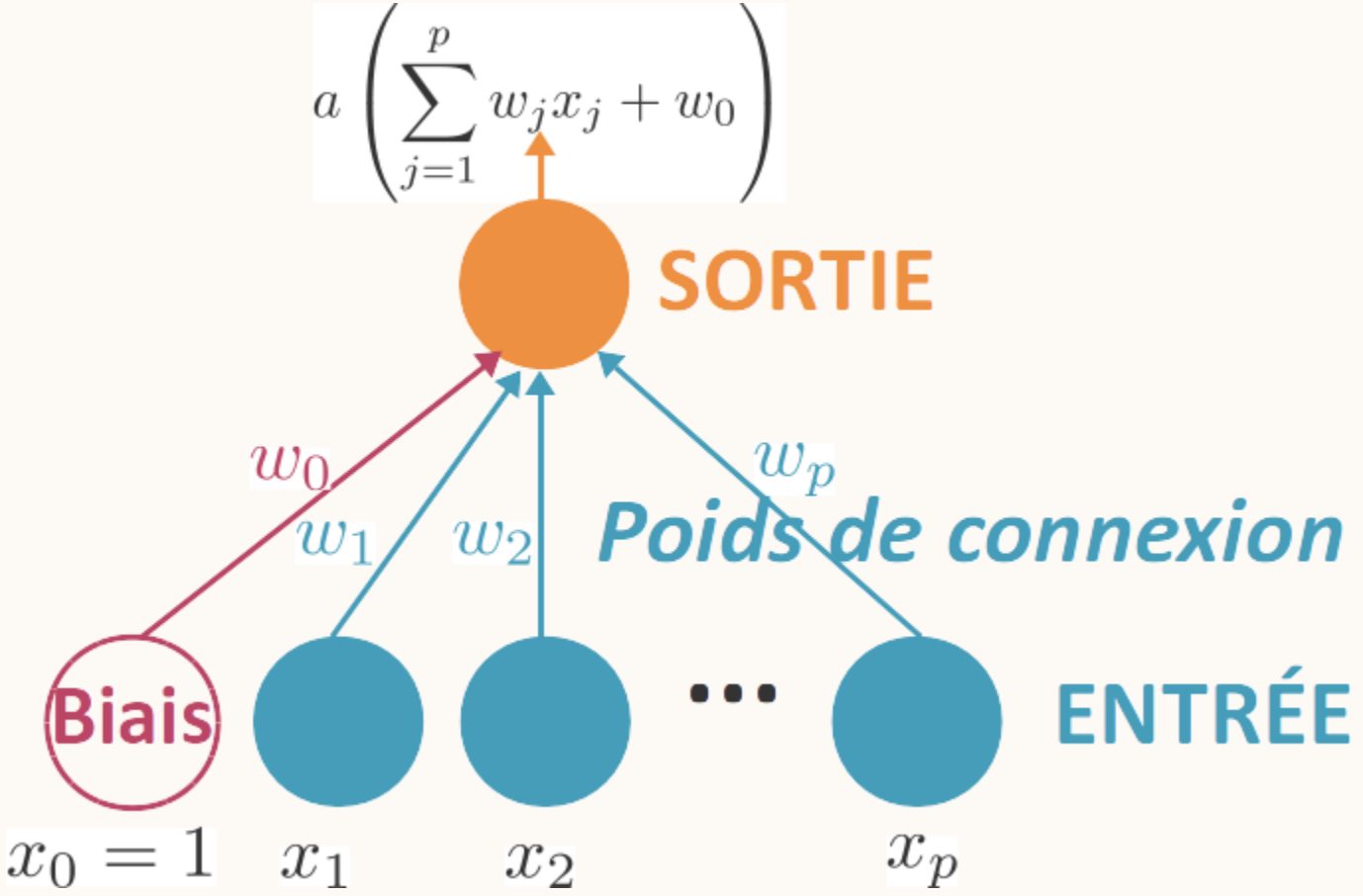
- Définition : réseaux de neurones



Constituants d'un réseau de neurones :

1) Couches de neurones(layers) dont la première est constituée des paramètres d'entrée et la dernière du résultat final
2) Neurones qui constituent les nœuds du réseau.
3) poids affectes à chaque arrêtes du réseau et qui définissent les liens entre les différents nœuds
4) Fonction d'activation qui permet d'activer les neurones



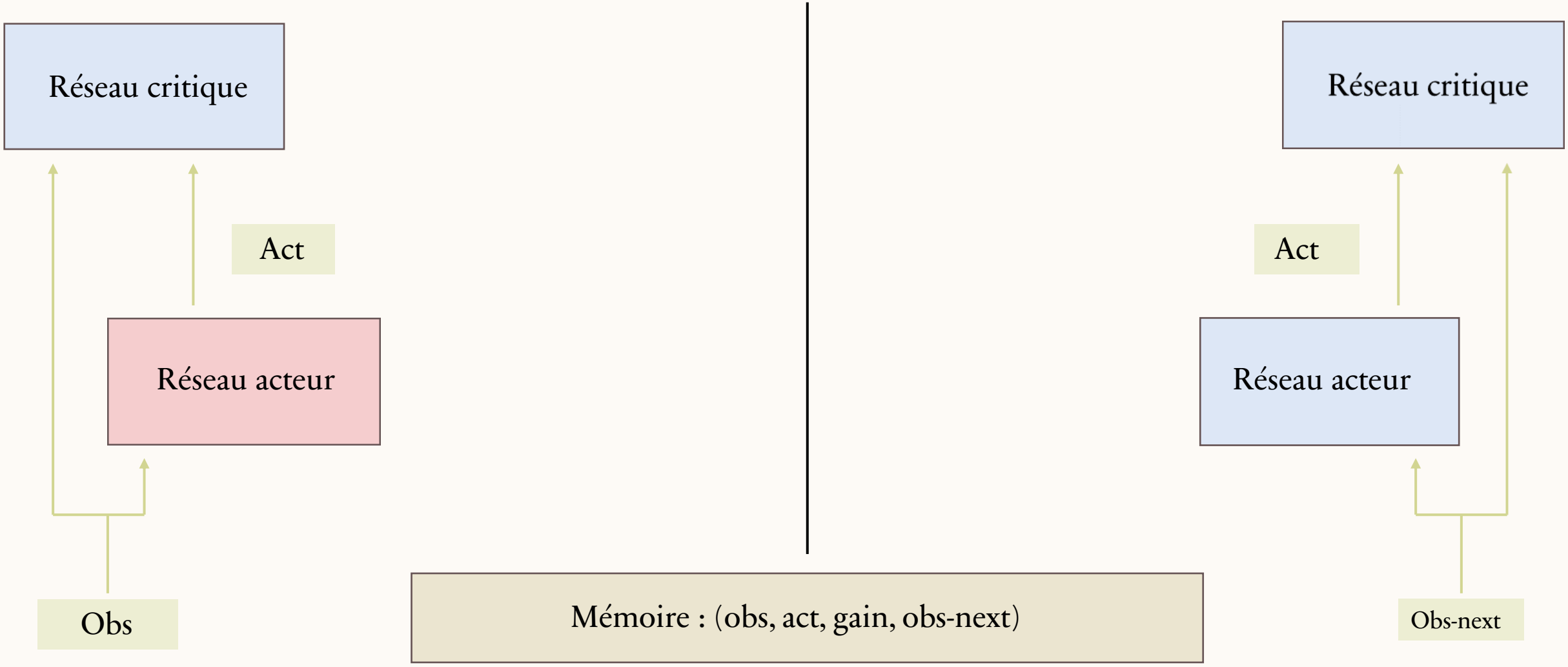


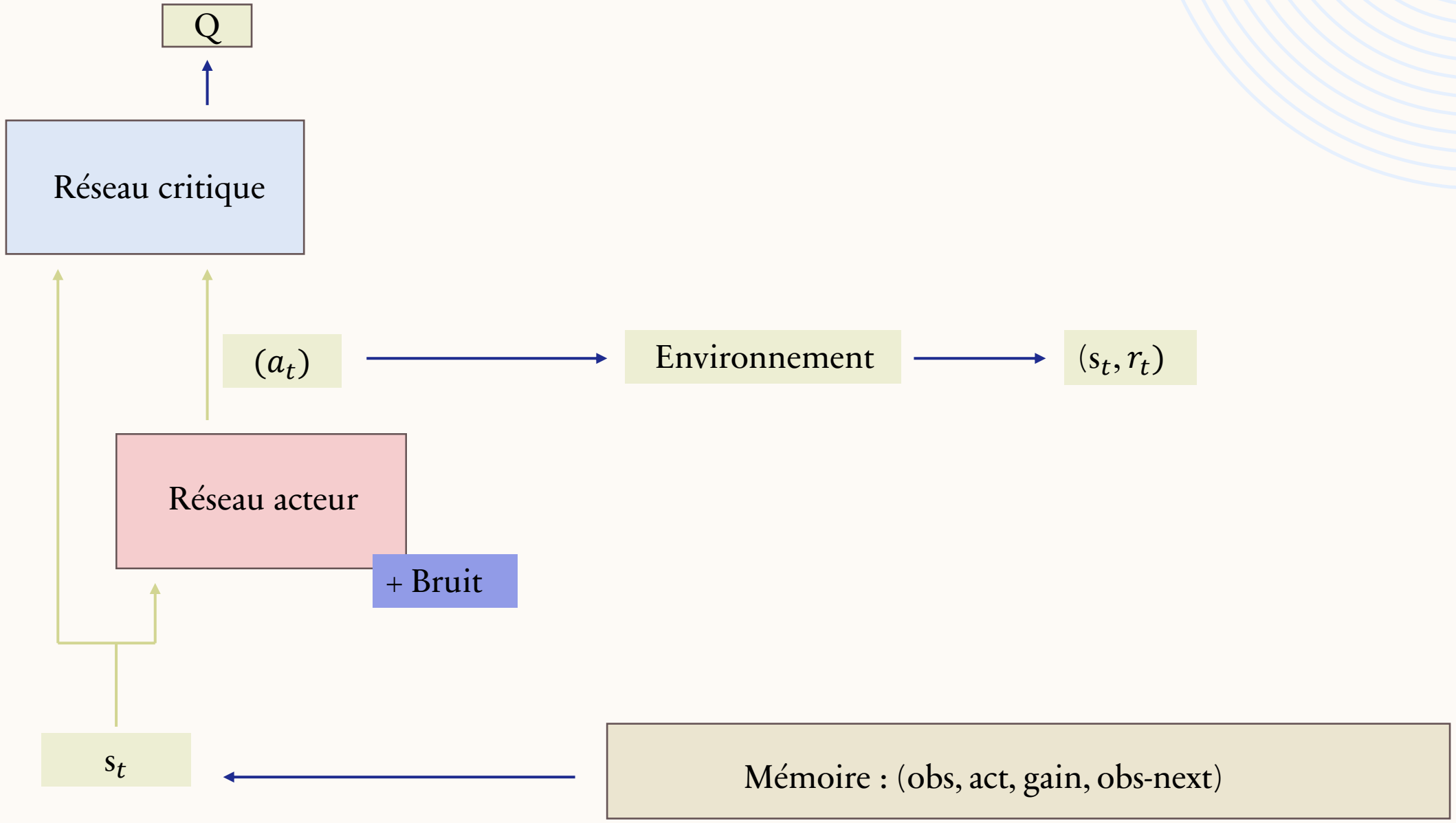
- $\forall l \; \forall i \;\; a_i^{(l)} = f\left(\sum_j w_{ij}^{(l)} \, a_j^{(l-1)} + b_i^{(l)}\right)$

COMMENT PERFECTIONNER ET RENDRE ACCESSIBLE À TOUS LE FES BIKE ?

- Programmes d'automatisation :Deep Deterministic Policy Gradients (DDPG)

Elements constituants cete méthode :



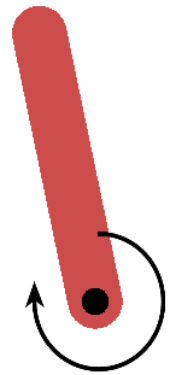

Q
Réseau critique
(a_t)
Environnement
(s_t, r_t)
Réseau acteur
+ Bruit
s_t
Mémoire : (obs, act, gain, obs-next)

Méthode d'apprentissage

- $a_{t+1} = \mu(s_{t+1}|\theta^{\mu}) + \text{bruit(t)}$
- $y_t = r_t + \gamma\, Q'\left(s_{t+1}, a_{t+1}|\theta^{Q'}\right)$
- $L(\theta^Q) = \frac{1}{N}\sum_1^N \left(y_i - \left(Q \quad \left(s_i, a_i|\theta^Q\right)\right)\right)^2$
- $J(\theta^Q) = \frac{1}{N}\sum_1^N \nabla_a Q(s,a|\theta^Q)\,|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
- $\theta^Q \leftarrow \theta^Q - \alpha\nabla_{\theta^Q} L(\theta^Q)$
- $\theta^\mu \leftarrow \tau\theta^\mu + (1-\tau)\theta^\mu$
- $\theta^{\mu'} \leftarrow \tau\theta^{\mu'} + (1-\tau)\theta^{\mu'}$

ETUDE PRATIQUE

Mise en équilibre d'un pendule



https://raw.githubusercontent.com/mediasia-labs/openai-gym-pendulum-v0/master/screenshot.png

PARAMÉTRAGES

Réseau critique :

. 1 couche entrée qui prend comme paramètre l'etat du système et l'action choisi par le réseau acteur.
. Chemin acteur : une couche de 32 neurones actives par la fonction ReLU
. Chemin critique : 2 couches de neurones de 16 et 32 neurones actives par la fonction ReLU
. Chemin commun : 2 couches de neurones cachés de 256 neurones activés avec la fonction ReLU

Réseaux acteurs :

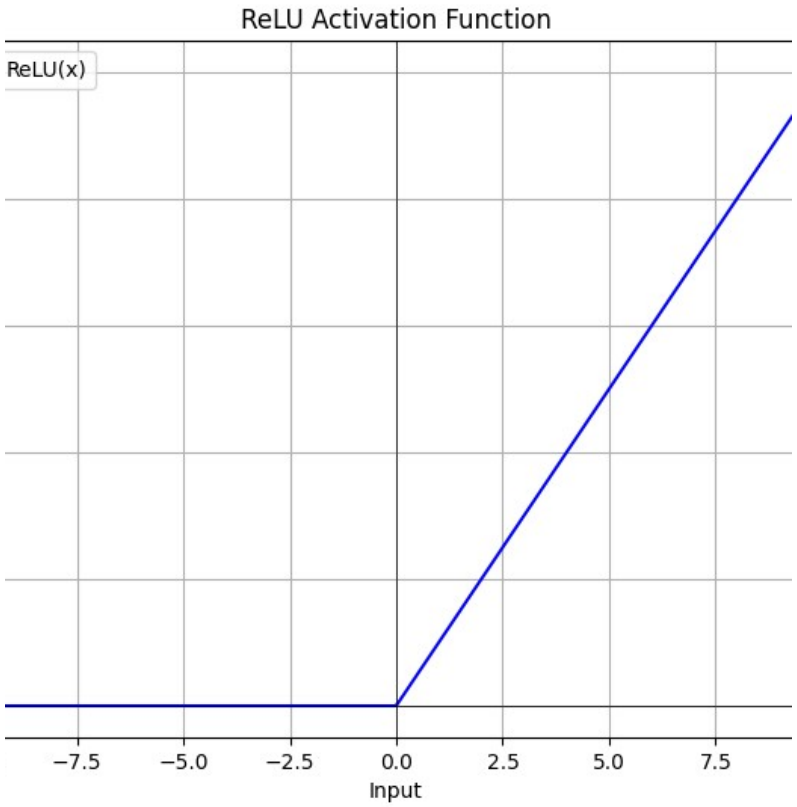
. 1 couche entrée qui prend comme paramètre l'etat du système
. 2 couches de neurones cachés de 256 neurones activés avec la fonction ReLU
. 1 couche sortie constituée de 1 neurone activée par la fonction Tanh

$$h_1 = ReLU(w_1 \quad + b_1 \;)$$
$$h_2 \quad = ReLU(w_2 \;\; h_1 + b_1 \;)$$
$$a = \tanh(w_3 h_2 + b_3)$$

$$h_{s1} \quad = ReLU(w_{s1} \quad + b_{s1} \;)$$
$$h_{s2} \quad = ReLU(w_{s2} \;\; h_{s1} \quad + b_{s2} \;)$$
$$h_a = ReLU(w_a a + b_a)$$
$$h_{c1} \quad = ReLU(w_{c1} \;\; [h_{s2}, h_a] + b_{c1} \;)$$
$$h_{c2} \quad = ReLU(w_{c2} \;\; h_{c1} + b_{c2} \;)$$

$$Q(s,a) = w_0 h_{c2} + b_0$$

Code qui affiche la fonction ReLU

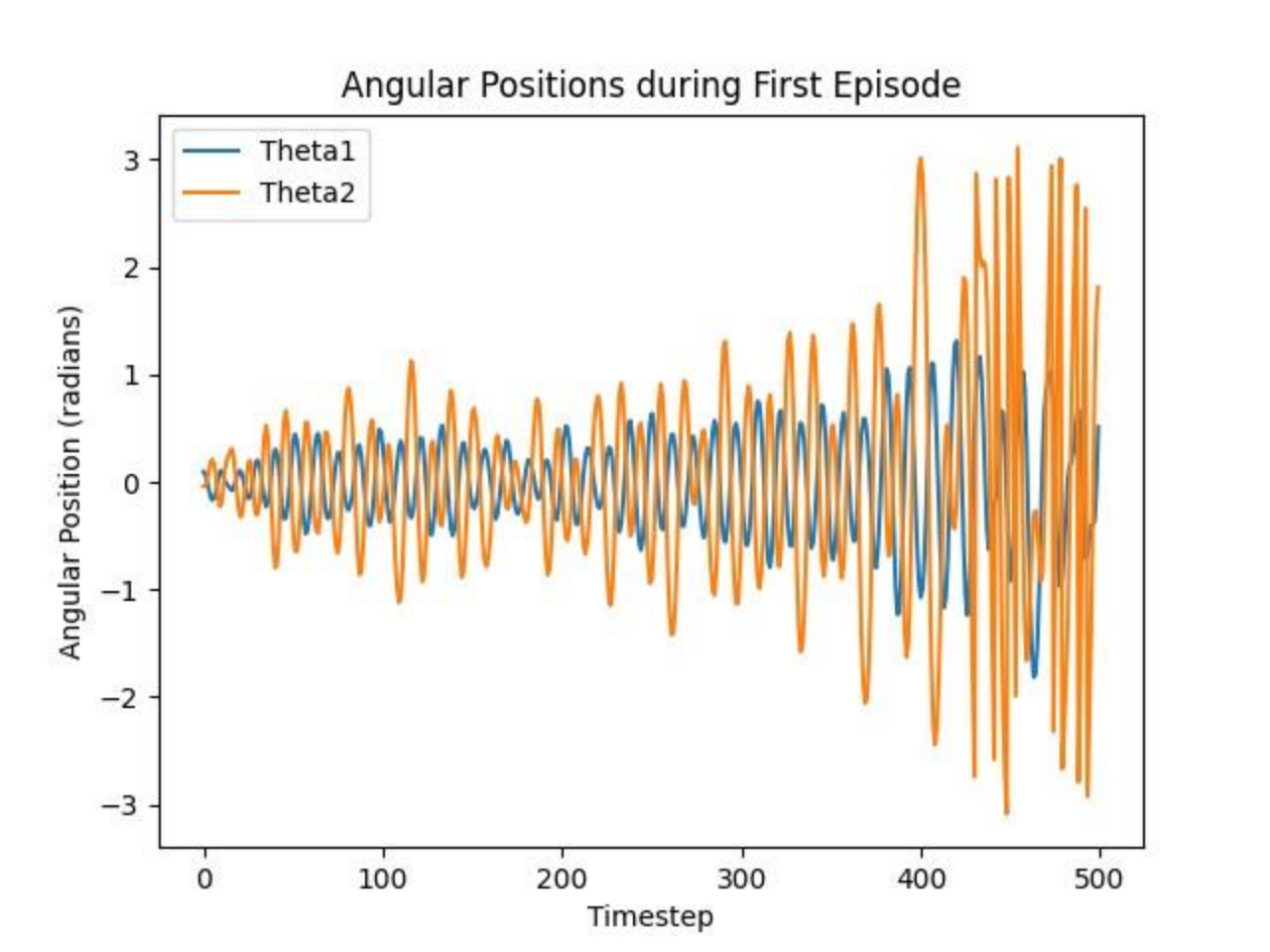


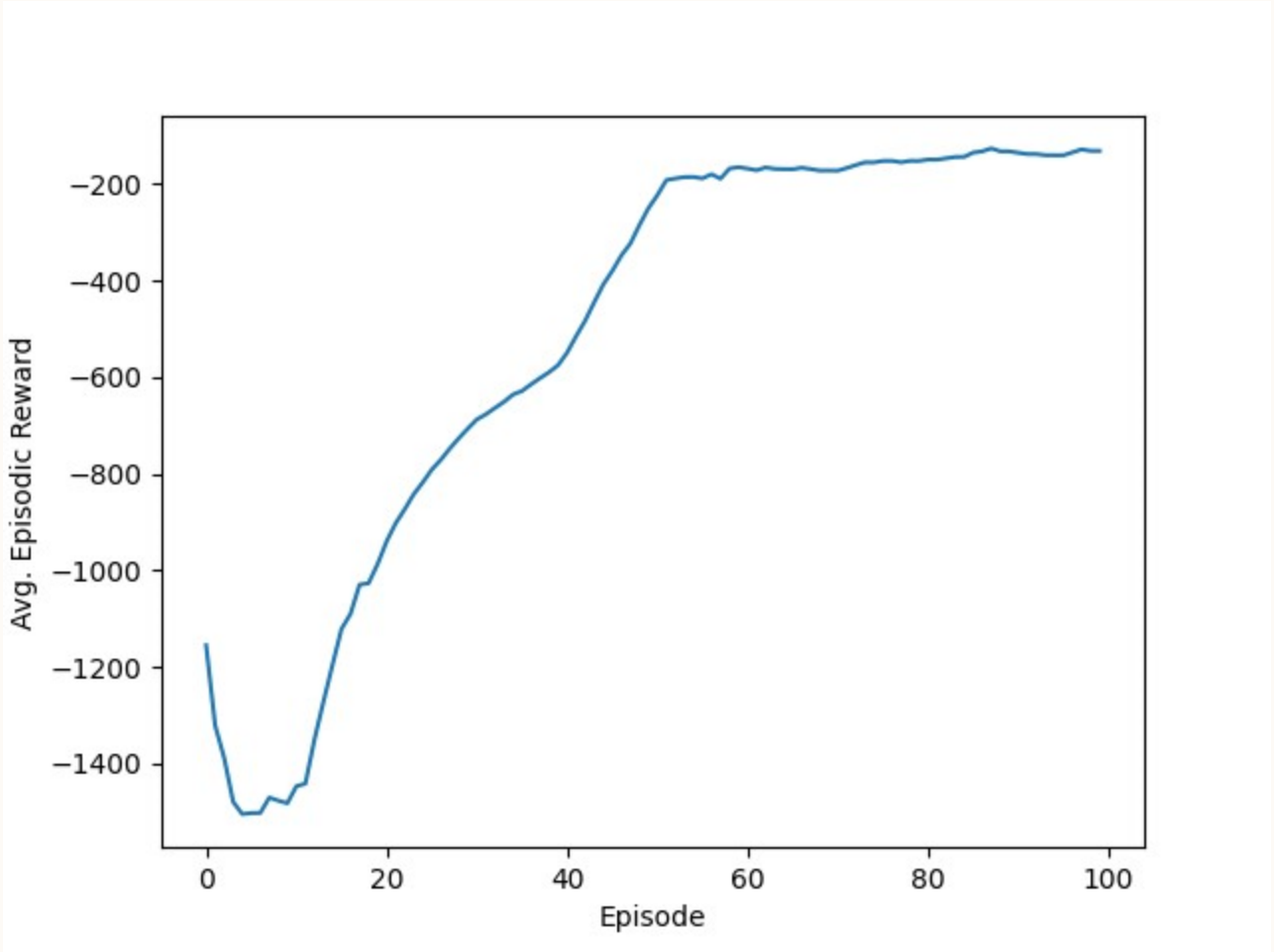

```python
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
    return np.maximum(0, x)

# Generate an array of values
x = np.linspace(-10, 10, 400)
y = relu(x)

# Plot the ReLU function
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='ReLU(x)', color='b')
plt.title('ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.legend()
plt.show()
```


Angular Positions during First Episode
Theta1
Theta2
Angular Position (radians)
3
2
1
0
−1
−2
−3
0
100
200
300
400
500
Timestep


Avg. Episodic Reward
−200
−400
−600
−800
−1000
−1200
−1400
0
20
40
60
80
100
Episode

CODE PYTHON

IMPORTATION DES BIBLIOTHÈQUES ET DE L'ENVIRONNEMENT

```python
import os
os.environ["KERAS_BACKEND"] = "tensorflow"

import keras
from keras import layers
import tensorflow as tf
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt

# Environment setup
env = gym.make("Pendulum-v1")

num_states = env.observation_space.shape[0]
num_actions = env.action_space.shape[0]
upper_bound = env.action_space.high[0]
lower_bound = env.action_space.low[0]
```

IMPLÉMENTATION DU PROCESSUS DE ORNSTEIN-UHLENBECK POUR LA CREATION DU BRUIT

```python
class OUActionNoise:
    def __init__(self, mean, std_deviation, theta=0.15, dt=1e-2, x_initial=None):
        self.theta = theta
        self.mean = mean
        self.std_dev = std_deviation
        self.dt = dt
        self.x_initial = x_initial
        self.reset()

    def __call__(self):
        x = (
            self.x_prev
            + self.theta * (self.mean - self.x_prev) * self.dt
            + self.std_dev * np.sqrt(self.dt) * np.random.normal(size=self.mean.shape)
        )
        self.x_prev = x
        return x

    def reset(self):
        if self.x_initial is not None:
            self.x_prev = self.x_initial
        else:
            self.x_prev = np.zeros_like(self.mean)
```

DÉFINITION DE LA MÉMOIRE (BUFFER)

```python
class Buffer:
    def __init__(self, buffer_capacity=100000, batch_size=64):
        self.buffer_capacity = buffer_capacity
        self.batch_size = batch_size
        self.buffer_counter = 0
        self.state_buffer = np.zeros((self.buffer_capacity, num_states))
        self.action_buffer = np.zeros((self.buffer_capacity, num_actions))
        self.reward_buffer = np.zeros((self.buffer_capacity, 1))
        self.next_state_buffer = np.zeros((self.buffer_capacity, num_states))

    def record(self, obs_tuple):
        index = self.buffer_counter % self.buffer_capacity
        self.state_buffer[index] = obs_tuple[0]
        self.action_buffer[index] = obs_tuple[1]
        self.reward_buffer[index] = obs_tuple[2]
        self.next_state_buffer[index] = obs_tuple[3]
        self.buffer_counter += 1
```

DÉFINITION DES RÉSEAUX ACTEUR ET CRITIQUE

```python
def get_actor():
    last_init = keras.initializers.RandomUniform(minval=-0.003, maxval=0.003)
    inputs = layers.Input(shape=(num_states,))
    out = layers.Dense(256, activation="relu")(inputs)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1, activation="tanh", kernel_initializer=last_init)(out)
    outputs = outputs * upper_bound
    model = keras.Model(inputs, outputs)
    return model

def get_critic():
    state_input = layers.Input(shape=(num_states,))
    state_out = layers.Dense(16, activation="relu")(state_input)
    state_out = layers.Dense(32, activation="relu")(state_out)
    action_input = layers.Input(shape=(num_actions,))
    action_out = layers.Dense(32, activation="relu")(action_input)
    concat = layers.Concatenate()([state_out, action_out])
    out = layers.Dense(256, activation="relu")(concat)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1)(out)
    model = keras.Model([state_input, action_input], outputs)
    return model
```

DÉFINITION DE LA POLITIQUE ET DES HYPERPARAMÈTRES

```
def policy(state, noise_object):
    sampled_actions = tf.squeeze(actor_model(state))
    noise = noise_object()
    sampled_actions = sampled_actions.numpy() + noise
    legal_action = np.clip(sampled_actions, lower_bound, upper_bound)
    return [np.squeeze(legal_action)]

std_dev = 0.2
ou_noise = OUActionNoise(mean=np.zeros(1), std_deviation=float(std_dev) * np.ones(1))

actor_model = get_actor()
critic_model = get_critic()
target_actor = get_actor()
target_critic = get_critic()
target_actor.set_weights(actor_model.get_weights())
target_critic.set_weights(critic_model.get_weights())
critic_lr = 0.002
actor_lr = 0.001
critic_optimizer = keras.optimizers.Adam(critic_lr)
actor_optimizer = keras.optimizers.Adam(actor_lr)
total_episodes = 100
gamma = 0.99
tau = 0.005
buffer = Buffer(50000, 64)
```

MISE À JOUR DES RÉSEAUX ACTEUR ET CRITIQUE

```python
@tf.function
def update(self, state_batch, action_batch, reward_batch, next_state_batch):
    with tf.GradientTape() as tape:
        target_actions = target_actor(next_state_batch, training=True)
        y = reward_batch + gamma * target_critic([next_state_batch, target_actions], train
        critic_value = critic_model([state_batch, action_batch], training=True)
        critic_loss = tf.math.reduce_mean(tf.math.square(y - critic_value))
    critic_grad = tape.gradient(critic_loss, critic_model.trainable_variables)
    critic_optimizer.apply_gradients(zip(critic_grad, critic_model.trainable_variables))

    with tf.GradientTape() as tape:
        actions = actor_model(state_batch, training=True)
        critic_value = critic_model([state_batch, actions], training=True)
        actor_loss = -tf.math.reduce_mean(critic_value)
    actor_grad = tape.gradient(actor_loss, actor_model.trainable_variables)
    actor_optimizer.apply_gradients(zip(actor_grad, actor_model.trainable_variables))

def learn(self):
    record_range = min(self.buffer_counter, self.buffer_capacity)
    batch_indices = np.random.choice(record_range, self.batch_size)
    state_batch = tf.convert_to_tensor(self.state_buffer[batch_indices])
    action_batch = tf.convert_to_tensor(self.action_buffer[batch_indices])
    reward_batch = tf.convert_to_tensor(self.reward_buffer[batch_indices])
    reward_batch = tf.cast(reward_batch, dtype="float32")
    next_state_batch = tf.convert_to_tensor(self.next_state_buffer[batch_indices])
    self.update(state_batch, action_batch, reward_batch, next_state_batch)
```

DÉFINITION DE LA BOUCLE D'ENTRAINEMENT

```
ep_reward_list = []
avg_reward_list = []

for ep in range(total_episodes):
    prev_state, _ = env.reset()
    episodic_reward = 0
    while True:
        tf_prev_state = tf.expand_dims(tf.convert_to_tensor(prev_state), 0)
        action = policy(tf_prev_state, ou_noise)
        state, reward, done, truncated, _ = env.step(action)
        buffer.record((prev_state, action, reward, state))
        episodic_reward += reward
        buffer.learn()
        update_target(target_actor, actor_model, tau)
        update_target(target_critic, critic_model, tau)
        if done or truncated:
            break
        prev_state = state
    ep_reward_list.append(episodic_reward)
    avg_reward = np.mean(ep_reward_list[-40:])
    print(f"Episode * {ep} * Avg Reward is ==> {avg_reward}")
    avg_reward_list.append(avg_reward)

plt.plot(avg_reward_list)
plt.xlabel("Episode")
plt.ylabel("Avg. Episodic Reward")
plt.show()

# Save the weights
actor_model.save_weights("pendulum_actor.weights.h5")
critic_model.save_weights("pendulum_critic.weights.h5")
target_actor.save_weights("pendulum_target_actor.weights.h5")
target_critic.save_weights("pendulum_target_critic.weights.h5")
```

MISE À JOUR DES RÉSEAUX PILOTE

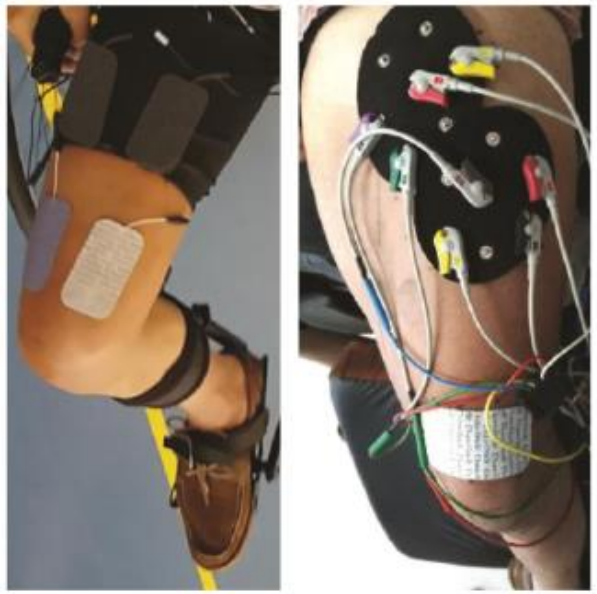
```
def update_target(target, original, tau):
    target_weights = target.get_weights()
    original_weights = original.get_weights()
    for i in range(len(target_weights)):
        target_weights[i] = original_weights[i] * tau + target_weights[i] * (1 - tau)
    target.set_weights(target_weights)
```

CONCLUSION

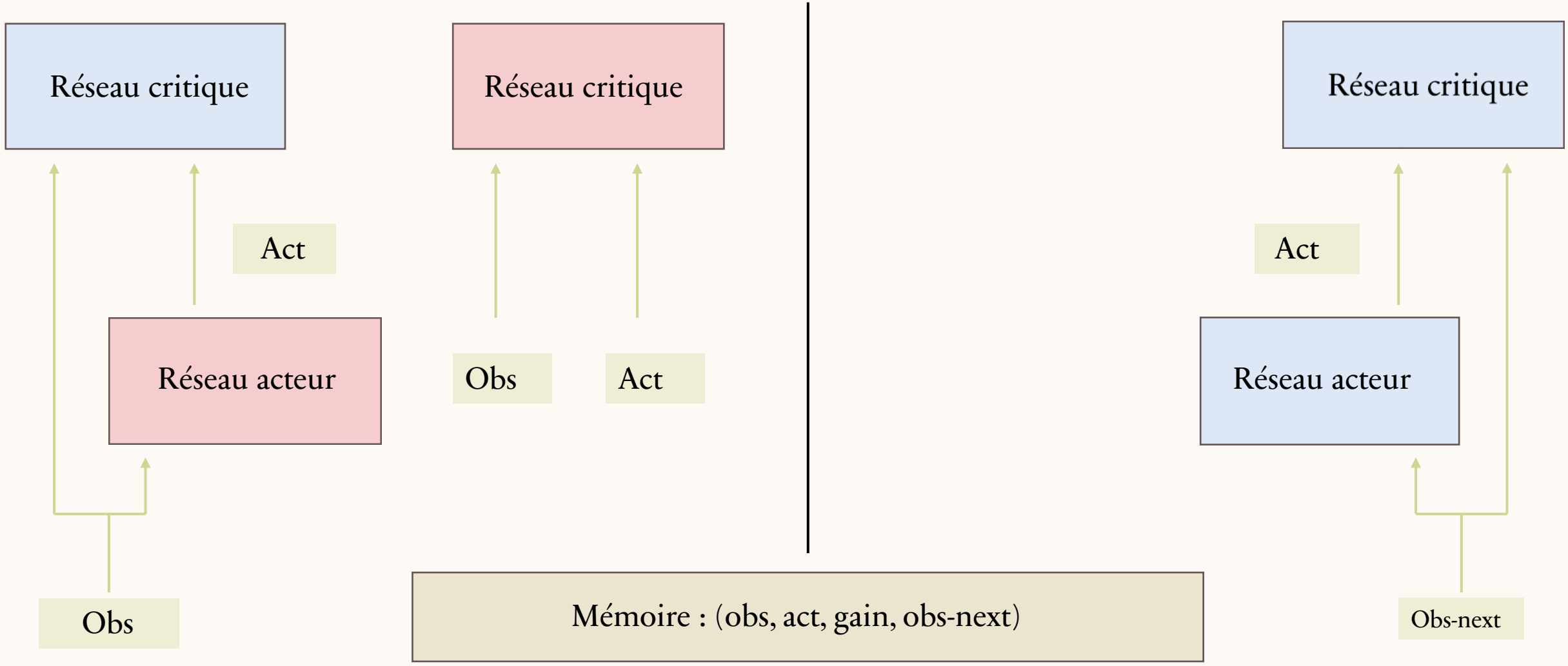
De nos jours l’utilisation de l’électrostimulation est de plus en plus commune cependant cette technologie est encore prometteuse et nécessite le travail d’ingénieurs et scientifiques notamment dans son utilisation en neurosciences appliques au monde du sport par exemple.

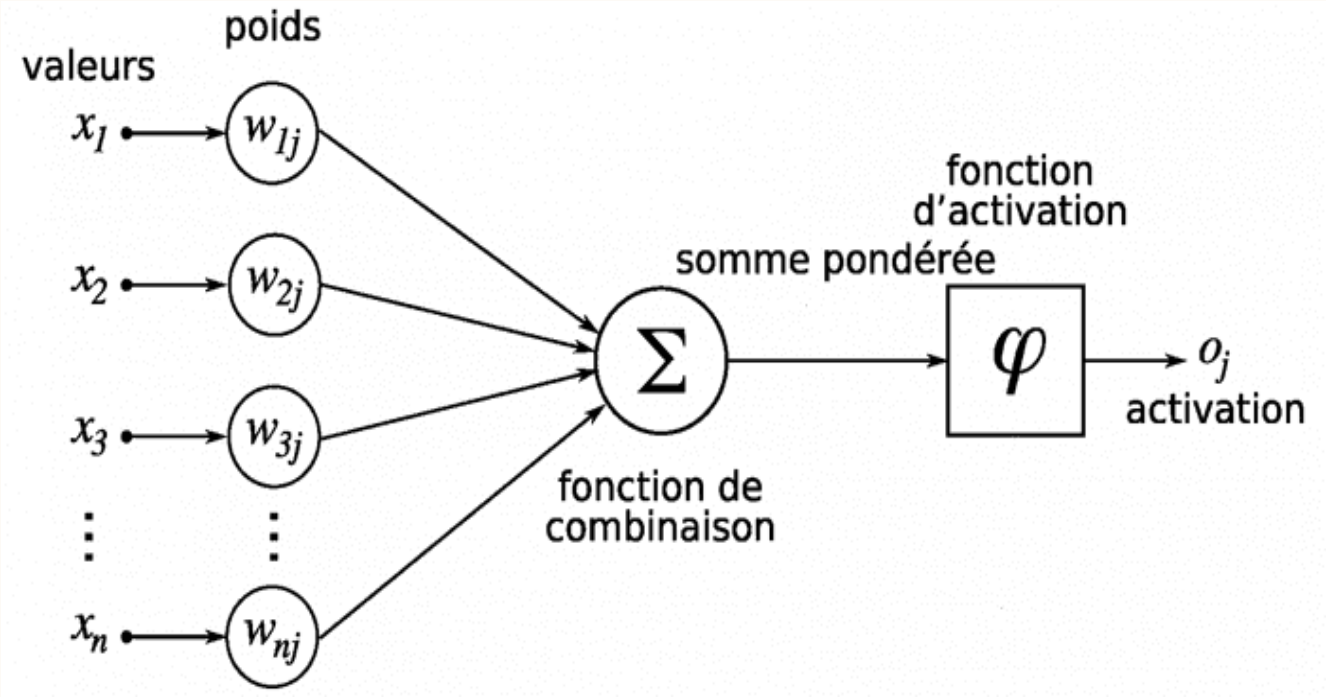
MERCI POUR VOTRE ATTENTION

Fig 3. Pedal orthoses: Berkelbike aluminum (left) and custom-made carbon (right).



Elements constituants cete méthode :





- $\forall l \; \forall i \;\; a_i^{(l)} = f\left(\sum_j w_{ij}^{(l)} \; a_j^{(l-1)} + b_i^{(l)}\right)$