Assignment 5: Neuroevolution

Goal: Implement an Evolutionary Algorithm to optimize an Artificial Neural Network (ANN) based controller for the CartPole task in OpenAI Gym environment.

CartPole evaluation environment functions are provided. Your goal is to implement your ANN to control the cartpole and use your Evolutionary Algorithm to optimize the ANN parameters (weights).

Please answer the `Questions` and implement coding `Tasks` by filling **PLEASE FILL IN** sections. *Documentation* of your code is also important. You can find the grading scheme in implementation cells.

- Plagiarism is automatically checked and set to **0 points**
- It is allowed to learn from external resources but copying is not allowed. If you use any external resource, please cite them in the comments (e.g. `# source: https://...../` (see `fitness_function`))

Install Prerequisites

In [1]:
```
# Run this cell to install the required libraries
%pip install numpy matplotlib scipy
```

```
Requirement already satisfied: numpy in c:\users\hghol\appdata\local\programs\python
\python311\lib\site-packages (1.26.4)
Requirement already satisfied: matplotlib in c:\users\hghol\appdata\local\programs\p
ython\python311\lib\site-packages (3.8.4)
Requirement already satisfied: scipy in c:\users\hghol\appdata\local\programs\python
\python311\lib\site-packages (1.13.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hghol\appdata\local\prog
rams\python\python311\lib\site-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hghol\appdata\local\programs
\python\python311\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hghol\appdata\local\pro
grams\python\python311\lib\site-packages (from matplotlib) (4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hghol\appdata\local\pro
grams\python\python311\lib\site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\hghol\appdata\local\progr
ams\python\python311\lib\site-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=8 in c:\users\hghol\appdata\local\programs\py
thon\python311\lib\site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hghol\appdata\local\prog
rams\python\python311\lib\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hghol\appdata\local
\programs\python\python311\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\hghol\appdata\local\programs\pyt
hon\python311\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Imports

In [2]:
```
# Necessary Libraries
import matplotlib.pyplot as plt
import numpy as np
%pip install moviepy
```

```
Requirement already satisfied: moviepy in c:\users\hghol\appdata\local\programs\pyth
on\python311\lib\site-packages (1.0.3)
Requirement already satisfied: decorator<5.0,>=4.0.2 in c:\users\hghol\appdata\local
\programs\python\python311\lib\site-packages (from moviepy) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in c:\users\hghol\appdata\local\pro
grams\python\python311\lib\site-packages (from moviepy) (4.66.2)
Requirement already satisfied: requests<3.0,>=2.8.1 in c:\users\hghol\appdata\local
\programs\python\python311\lib\site-packages (from moviepy) (2.31.0)
Requirement already satisfied: proglog<=1.0.0 in c:\users\hghol\appdata\local\progra
ms\python\python311\lib\site-packages (from moviepy) (0.1.10)
Requirement already satisfied: numpy>=1.17.3 in c:\users\hghol\appdata\local\program
s\python\python311\lib\site-packages (from moviepy) (1.26.4)
Requirement already satisfied: imageio<3.0,>=2.5 in c:\users\hghol\appdata\local\pro
grams\python\python311\lib\site-packages (from moviepy) (2.34.1)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in c:\users\hghol\appdata\local
\programs\python\python311\lib\site-packages (from moviepy) (0.4.9)
Requirement already satisfied: pillow>=8.3.2 in c:\users\hghol\appdata\local\program
s\python\python311\lib\site-packages (from imageio<3.0,>=2.5->moviepy) (10.3.0)
Requirement already satisfied: setuptools in c:\users\hghol\appdata\local\programs\p
ython\python311\lib\site-packages (from imageio-ffmpeg>=0.2.0->moviepy) (65.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\hghol\appdata\lo
cal\programs\python\python311\lib\site-packages (from requests<3.0,>=2.8.1->moviepy)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hghol\appdata\local\programs
\python\python311\lib\site-packages (from requests<3.0,>=2.8.1->moviepy) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hghol\appdata\local\pr
ograms\python\python311\lib\site-packages (from requests<3.0,>=2.8.1->moviepy) (2.2.
1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hghol\appdata\local\pr
ograms\python\python311\lib\site-packages (from requests<3.0,>=2.8.1->moviepy) (202
4.2.2)
Requirement already satisfied: colorama in c:\users\hghol\appdata\local\programs\pyt
hon\python311\lib\site-packages (from tqdm<5.0,>=4.11.2->moviepy) (0.4.6)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
# Enables inline matplotib graphs
%matplotlib inline
# Comment the line above and uncomment the lines below to have interactive plots
# WARN: may cause dependency issues
# %matplotlib qt5
# %pip install PyQt5
# plt.ion()
```

In [4]:
```
%pip install gymnasium
import gymnasium as gym
%pip install gymnasium[classic-control]
```

```
Requirement already satisfied: gymnasium in c:\users\hghol\appdata\local\programs\py
thon\python311\lib\site-packages (0.29.1)
Requirement already satisfied: numpy>=1.21.0 in c:\users\hghol\appdata\local\program
s\python\python311\lib\site-packages (from gymnasium) (1.26.4)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\hghol\appdata\local\pr
ograms\python\python311\lib\site-packages (from gymnasium) (3.0.0)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\hghol\appdata\lo
cal\programs\python\python311\lib\site-packages (from gymnasium) (4.10.0)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\hghol\appdata
\local\programs\python\python311\lib\site-packages (from gymnasium) (0.0.4)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: gymnasium[classic-control] in c:\users\hghol\appdata
\local\programs\python\python311\lib\site-packages (0.29.1)Note: you may need to res
tart the kernel to use updated packages.

Requirement already satisfied: numpy>=1.21.0 in c:\users\hghol\appdata\local\program
s\python\python311\lib\site-packages (from gymnasium[classic-control]) (1.26.4)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\hghol\appdata\local\pr
ograms\python\python311\lib\site-packages (from gymnasium[classic-control]) (3.0.0)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\hghol\appdata\lo
cal\programs\python\python311\lib\site-packages (from gymnasium[classic-control])
(4.10.0)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\hghol\appdata
\local\programs\python\python311\lib\site-packages (from gymnasium[classic-control])
(0.0.4)
Requirement already satisfied: pygame>=2.1.3 in c:\users\hghol\appdata\local\program
s\python\python311\lib\site-packages (from gymnasium[classic-control]) (2.5.2)
```

Question 1 (0-0.25-0.5 pt): Following link provides more information about the CartPole environemnt we would like to find an ANN to control:
https://www.gymlibrary.dev/environments/classic_control/cart_pole/

Please have a look at the link and note the observation and action spaces, how many dimensions they have? Are they continous or discrete, and what kinds of value they can get?

Answer: The action and observation spaces have 2 and 4 dimensions, respectively.

The observation space has cart position, cart velocity, pole angle, and pole angular velocity, and it is continuous. This can have any real numbers within the provided range. The observation space contains the range of inputs that are then used in the rest of the neural network. The range of possible options must be known before the code is executed.

The action space contains one dimension, with two value choices: cart pushed left and cart pushed right, and it is discrete. It can take values of 0 or 1, which represent whether the action should be applied or not. The action space is the set of possible options that can be performed following implementation of the algorithm. It needs to know the set of possible options before executing.

Question 2 (0-0.25-0.5 pt): What is your proposed ANN architecture and why? Please also discuss the activation functions you choose.

Answer: My proposed ANN architecture consists of an input layer, hidden layer, and output layer. The input layer has 4 neurons, which account for the four observation parameters described in the previous question. The hidden layer contains 8 neurons, as this gives the algorithm enough capacity to learn the relationships between the input and output layer without adding too much complexity to the algorithm. I chose this number because it seemed like a good balance and did not overload my runtime. The output layer has two nuerons to account for the two action options of the network: left or right.

Activation functions:

tanh (hyperbolic tangent): This was used to normalize the output of my hidden layer, as it outputs values between -1 and 1. It also addresses the vanishing gradient problem and captures non-linear relationships quite well.

linear activation in output layer: This was used because the outputs are directly used to determine the next action using argmax. Thus, linear activation was appropriate since it compares the values and outputs raw values which can then be used to decide on the resulting action.

Task 1: Implementation of Evolutionary Algrotihm (0-1.6-3.8-4.2-5 pt): Implement your evolutionary algorithm to find an ANN controller for the CartPole task.

In [72]:
```
###################################
# Grading
# 0 pts if the code does not work, code works but it is fundamentally incorrect
# 1.6 pts if the code works but some functions are incorrect and it is badly explai
# 3.8 pts if the code works but some functions are incorrect but it is explained we
# 4.2 pts if the code works very well aligned with the task without any mistakes, b
# 5 pts if the code works very well aligned with the task without any mistakes, and
###################################################################


# Artificial Neural Network parameters (weights)
# See here: https://www.gymlibrary.dev/environments/classic_control/cart_pole/ for
# PLEASE SPECIFY BELOW
inp = 4 # Number of input neurons
hid = 8 # Number of hidden neurons
out = 2 # Nubmer of output neurons
####################


# input_weights = np.random.uniform(low=-0.1, high=0.1, size=(inp, hid))
# hidden_weights = np.random.uniform(low=-0.1, high=0.1, size=(hid, out))

#Open AI gym environment
env = gym.make("CartPole-v1")
```

```
# CartPole evaluation function
def cartpole(x):

    ###########
    # PLEASE FILL IN
    # Hint: x is an individual in evolutionary algorithms and needs to map to the c

    # input_weights = np.random.rand(inp, hid)
    # hidden_weights = np.random.rand(hid, out)
    #########
    input_weights, hidden_weights=x
    # Reset environment
    observation, info = env.reset(seed = 0)

    rew = 0 # Initial reward
    step = 0; #step counter
    done = False
    maxStep = 1000  # maximum number of steps
    while not done and step<1000:




      ###################
        hidden_layer_activation = np.dot(observation, input_weights) #dot product o
        hidden_layer_output = np.tanh(hidden_layer_activation) # tanh to normalize
        output_layer_activation = np.dot(hidden_layer_output, hidden_weights) # dot

        # Determine action based on the sign of the output
        #check for larger - argmax
        action = np.argmax(output_layer_activation)

        # If the value at position [0] is greater, action should be 0, otherwise 1
        if output_layer_activation[0] > output_layer_activation[1]:
            action = 0
        else:
            action = 1
      # PLEASE FILL IN
      # Hint: Provide input to ANN and find the output to be the action
      # action = ?


      # action should be provided based on the output of the artifial neural networ
        observation, reward, done, tr, info = env.step(action)
        step+=1 # step counter
        rew = rew + reward # after each step increment reward

    env.close()
    return np.minimum(maxStep, rew) # return the reward or maxStep (if maxStep < 10



#CartPole evaluation function with video recording
def cartpole_record_video(x):
    # tmp_env = gym.make("CartPole-v1", render_mode="rgb_array")
```

```
# # Video recording function - be sure to check the folder path - you should se
# env = gym.wrappers.RecordVideo(env=tmp_env, video_folder="content/video/cartp

# ###########
# # PLEASE FILL IN
# # Hint: x is an individual in evolutionary algorithms and needs to map to the
# # input_weights = np.random.rand(inp, hid)
# # hidden_weights = np.random.rand(hid, out)
# # #########

# # Reset environment
# observation, info = env.reset(seed = 0)

# rew = 0 # Initial reward
# step = 0; #step counter
# done = False
# maxStep = 1000  # maximum number of steps
# while not done and step<1000:

#     # hidden_layer_output=observation
#     # for i in range (hid):
#     #     hidden_layer_output = np.dot(x, hidden_layer_output)
#     #     if i ! = hid(1):


#   ####################
#     hidden_layer_activation = np.dot(observation, input_weights)
#     hidden_layer_output = np.tanh(hidden_layer_activation)
#     output_layer_activation = np.dot(hidden_layer_output, hidden_weights)

#     # Determine action based on the sign of the output
#     #check for larger - argmax
#     action = np.argmax(output_layer_activation)

#     # # If the value at position [0] is greater, action should be 0, otherwis
#     # if output_layer_activation[0] > output_layer_activation[1]:
#     #     action = 0
#     # else:
#     #     action = 1
# #########


# # Reset environment
# observation, info = env.reset(seed = 0)

# env.start_video_recorder()


# rew = 0 # Initial reward
# step = 0; #step counter
# done = False
# maxStep = 1000  # maximum number of steps
# while not done and step<1000: # run nStep number of time

#   ####################
```

```
    #   # PLEASE FILL IN
    #   # Hint: Provide input to ANN and find the output to be the action
    #   # action = ?



    #   # action should be provided based on the output of the artifial neural netw
    #   observation, reward, done, tr, info = env.step(action)
    #   step+=1 # step counter
    #   rew = rew + reward # after each step increment reward
    #   env.render()

    # env.close_video_recorder()
    # env.close()
    # return np.minimum(maxStep, rew) # return the reward or maxStep (if maxStep <

    tmp_env = gym.make("CartPole-v1", render_mode="rgb_array")
    env = gym.wrappers.RecordVideo(env=tmp_env, video_folder="content/video/cartpol

    input_weights, hidden_weights = x

    # Reset environment
    observation, info = env.reset(seed=0)

    rew = 0  # Initial reward
    step = 0  # Step counter
    done = False
    maxStep = 1000  # Maximum number of steps

    while not done and step < 1000:
        hidden_layer_activation = np.dot(observation, input_weights) #matrix dot pr
        hidden_layer_output = np.tanh(hidden_layer_activation) #tanh normalization
        output_layer_activation = np.dot(hidden_layer_output, hidden_weights) #matr

        action = np.argmax(output_layer_activation) #taking maximize of values to d
        observation, reward, done, tr, info = env.step(action)
        step += 1
        rew += reward

    # Close the video recorder and environment properly
    env.close()
    tmp_env.close()
    return np.minimum(maxStep, rew)


# CartPole evaluation function for visualizing the cartpole environment
def cartpole_visualize(x):
    tmp_env = gym.make("CartPole-v1", render_mode="human")

    ###########
    # PLEASE FILL IN
    # Hint: x is an individual in evolutionary algorithms and needs to map to the c
    input_weights = np.random.rand(inp, hid)
    hidden_weights = np.random.rand(hid, out)
    #########
```

```
    # Reset environment
    observation, info = env.reset(seed = 0)

    rew = 0 # Initial reward
    step = 0; #step counter
    done = False
    maxStep = 1000  # maximum number of steps
    while not done and step<1000:

        # hidden_layer_output=observation
        # for i in range (hid):
        #     hidden_layer_output = np.dot(x, hidden_layer_output)
        #     if i ! = hid(1):


      ####################
        hidden_layer_activation = np.dot(observation, input_weights) #dot product o
        hidden_layer_output = np.tanh(hidden_layer_activation) # normalization of m
        output_layer_activation = np.dot(hidden_layer_output, hidden_weights) #dot

        # Determine action based on the sign of the output
        #check for larger - argmax
        action = np.argmax(output_layer_activation)

        # If the value at position [0] is greater, action should be 0, otherwise 1
        if output_layer_activation[0] > output_layer_activation[1]:
            action = 0
        else:
            action = 1
    #########

    # Reset environment
    observation, info = tmp_env.reset(seed = 0)

    rew = 0 # Initial reward
    step = 0; #step counter
    done = False
    maxStep = 1000  # maximum number of steps
    while not done and step<1000: # run nStep number of time

      ####################
      # PLEASE FILL IN
      # Hint: Provide input to ANN and find the output to be the action
      # action = ?

      # action should be provided based on the output of the artifial neural networ
      observation, reward, done, tr, info = tmp_env.step(action)
      step+=1 # step counter
      rew = rew + reward # after each step increment reward
      tmp_env.render()

    tmp_env.close()
    return np.minimum(maxStep, rew) # return the reward or maxStep (if maxStep < 10
###code from assignment 2###

def initialization(population_size):
```

```
    population=[] #array of weights
    for _ in range(population_size):
    #iterates through entire population and adds weights to x array
        input_weights = np.random.uniform(low=-0.1, high=0.1, size=(inp, hid))
        hidden_weights = np.random.uniform(low=-0.1, high=0.1, size=(hid, out))
        #weights are randomly generated based on input, hidden, and output layer pa
        x = (input_weights, hidden_weights)
        population.append(x)
        #this generates a random population that adheres to the provided dimensions


    return population #return population


# Implement the evaluation function that can evaluate all the solutions in a given
def evaluation(x, objective_function):

    fitness=[]

    for individual in x:
        fitness.append(objective_function(individual))
    #creates array of possible solutions

    return fitness


# Implement the crossover operator by choosing a suitable method. For inspiration,
def crossover(x_parents, p_crossover):

    offspring = []
    x_parents=x_parents.tolist() #converts x_parents to a list




    for _ in range(num_parents // 2):
        parent1 = parents[np.random.randint(num_parents)]
        parent2 = parents[np.random.randint(num_parents)]
#uses crossover method to determine parents and children
        input_weights1, hidden_weights1 = parent1
        input_weights2, hidden_weights2 = parent2

        child_input_weights = input_weights1.copy()
        child_hidden_weights = hidden_weights1.copy()

        if np.random.rand() < p_crossover:
            crossover_point = np.random.randint(1, inp * hid)
            child_input_weights = np.concatenate((input_weights1.flatten()[:crossov
                                                  input_weights2.flatten()[crossove
        #randomly selects crossover point, then uses child weights to determine new
        if np.random.rand() < p_crossover:
            crossover_point = np.random.randint(1, hid * out)
            child_hidden_weights = np.concatenate((hidden_weights1.flatten()[:cross
                                                   hidden_weights2.flatten()[crosso
```

```
        offspring.append((child_input_weights, child_hidden_weights))
    ##################################################################

    return offspring



# Implement the crossover operator by choosing a suitable method. For inspiration,
def mutation(x, mutation_rate):

    input_weights, hidden_weights = x #sets weights equal to x value

    if np.random.rand() < mutation_rate:
        input_weights += np.random.uniform(low=-0.1, high=0.1, size=input_weights.s

    if np.random.rand() < mutation_rate:
        hidden_weights += np.random.uniform(low=-0.1, high=0.1, size=hidden_weights
#determines if mutation should occur or not based on randomly generated values

    return (input_weights, hidden_weights)

    # for trait in range(len(x)):
    #     prob=np.random.rand() # computes random probability
    #     if prob<=mutation_rate: # compares probability to mutation rate to see if
    #         x[trait]=np.random.rand() # randomly mutates a trait

    # ##################################################################

    # return x


def parent_selection(x, f):

    x=x.tolist() #converts x to a list
    num_parents=int(len(x) / 2) # divide length of x by two to get parent options
    total_sort=sorted(zip(f,x)) # sorts parent list by fitness
    sorted_x=[x for _, x in total_sort]

    x_parents=sorted_x[-num_parents:] # selects parent from sorted list
    f_parents=f[-num_parents:] # analyzes fitness values of parent options
    x_parents=np.array(x_parents) # adds parents to array of x_parents

    ####################################
    return x_parents, f_parents


def survivor_selection(x, f, x_offspring, f_offspring):
    """Select the survivors, for the population of the next generation"""

    x=x.tolist()
    x_offspring=x_offspring.tolist()
    # x and x_offspring converted to lists to be able to add togther

    x_total=x+x_offspring # total population
    f_total=f+f_offspring # total fitness values
    num_survivors=len(x) # survivors needed
```

```
    sorted_x=[x for _, x in sorted(zip(f_total, x_total))] # sorted list based on f
    x=sorted_x[-num_survivors:] #orders list from highest to lowest values
    f=f_total[-num_survivors:]
    x=np.array(x) #array of selected survivors

    return x,f


    ######################################################################
######


# Implement your Evolutionary Algorithm to find the ANN weigths that can balance th
# Feel free to add any functions, such as initialization, crossover, etc.. to make

##EA algorithm is adapted from Assignmnent 2 to reflect 2x2 matrix form, maximizing
def ea(
    # hyperparameters of the algorithm
    population_size,
    max_fit_evals,  # Maximum number of evaluations
    p_crossover,  # Probability of performing crossover operator
    m_rate,  # mutation rate
    objective_function,  # objective function to be minimized
    #num_dimensions,
):
    ####################
    # PLEASE FILL IN

    # Calculate the maximum number of generations
    # Maximum number of function evaluations should be the same independent of the
    max_generations = int(max_fit_evals / population_size)  # DO NOT CHANGE



    ######################################################################
    # PLEASE FILL IN
    x = initialization(population_size) #initialization of function
    x_best=[]
    f_best=[]
    f=evaluation(x, objective_function) #function evaluation of fitness
    ######################################################################

    # Find the best individual and append to a list to keep track in each generatio
    idx = np.argmax(f)
    x_best = [x[np.argmax(f)]]
    f_best = [np.max(f)]

    # Loop over the generations
    for _ in range(max_generations - 1):
        # Perform the EA steps


        ######################################################################

        # PLEASE FILL IN
```

```
        f = evaluation(x, objective_function) #function evaluation definition
        parents, parents_fitness = parent_selection(x,f) #parents are selected
        offspring = crossover(parents, p_crossover) #uses crossover method on offsp
        offspring = mutation(offspring, m_rate) #uses mutation method on offspring
        offspring_fitness = evaluation(offspring, objective_function) #computes fit
        x, f = survivor_selection(x, f, offspring, offspring_fitness) #selects surv
        ###################################################################


        # Find the best individual in current generation and add to the list
        idx = np.argmax(f)
        xi_best = x[idx]
        fi_best = f[idx]
        if fi_best < f_best[-1]:
            x_best.append(xi_best)
            f_best.append(fi_best)
        else:
            x_best.append(x_best[-1])
            f_best.append(f_best[-1])




    # Hint: your implementation of your evolutionary algorithm
    # You may use the code you previously implemented during the course

    return x_best, f_best  # return the best solution (ANN weights) and the fitness
```

Check Your Implementation: Running The Evolutionary Algorithm

Run the cell below, if you implemented everything correctly, you should see the algorithm running. Furthermore,

In [38]:
```
# Dummy parameters, please add or remove based on your implementation
kwargs = {
    "population_size": 1000,
    "max_fit_evals": 1000,  # maximum number of fitness evaluations
    "p_crossover": 0.9,  # crossover probability
    "m_rate": 0.05,  # mutation rate, previously 0.1
    "objective_function": cartpole,
    #"num_dimensions": 10,
}
# Run your algorithm once and find the best ANN weigths found
env = gym.make("CartPole-v1")
x_best, f_best = ea(**kwargs)


# Print the best ANN weigths found and best fitness
print("Best ANN parameters found:",x_best[-1])
print("Best fitnes found:",f_best[-1])


# Evaluate your ANN weights again and record the video
if f_best[-1] >= 1000:
  cartpole_record_video(x_best[-1] )
```

```
  #or cartpole_visualize(x_best[-1] )
else:
  print("The best fitness 1000 was not found, try again!!")
```

```
Best ANN parameters found: (array([[ 0.03402866,  0.02304855,  0.0938808 , -0.043504
88, -0.03875568,
         0.00992052, -0.03455357, -0.04177599],
       [ 0.02119102,  0.07367793, -0.09404815,  0.06430077, -0.03301562,
         0.05794261, -0.06290838, -0.08160841],
       [-0.07199318, -0.08962079,  0.04910696, -0.06603492,  0.06714754,
         0.04634327,  0.03781677,  0.03406412],
       [ 0.0687214 ,  0.02320319,  0.03295476,  0.05190229,  0.06325197,
         0.08989176, -0.08380241,  0.06149431]]), array([[-0.02267759, -0.07131169],
       [ 0.00921187,  0.05239381],
       [ 0.06223871,  0.09660451],
       [-0.02245389, -0.00829576],
       [-0.09015526,  0.06321394],
       [-0.09803822, -0.02490035],
       [-0.00654624, -0.01785945],
       [ 0.05602028, -0.00717138]]))
Best fitnes found: 1000.0
Moviepy - Building video C:\Users\hghol\Downloads\content\video\cartpole\cartpole-ep
isode-0.mp4.
Moviepy - Writing video C:\Users\hghol\Downloads\content\video\cartpole\cartpole-epi
sode-0.mp4
```

```
Moviepy - Done !
Moviepy - video ready C:\Users\hghol\Downloads\content\video\cartpole\cartpole-episo
de-0.mp4
```

Question 3 (0-0.25-0.5 pt): Please comment on the behavior of the final solution. Were you able to find the best solution (i.e. ANN weights which achieves best fitness: 1000) and was it possible to controll the CartPole task without letting the the pole fall?

Answer: I was able to find the best solution, where the fitness was 1000 for every trial run. It was possible to control the CartPole task without letting the pole fall, which I was able to see in my video.

Average results of your algorithm

Remember that the EAs are sthocastic algorithms that can produce different results as a result of independent runs.

Therefore, we would like to see the average results and standard deviations.

Task 2 (0-1.5-3 pt): Please run your algorithm for at least 10 times and plot the average results and standard deviations. Below, you may add as many cells as you need for this

implementation and plot functions. You may use previous code you have developed/used during the course.

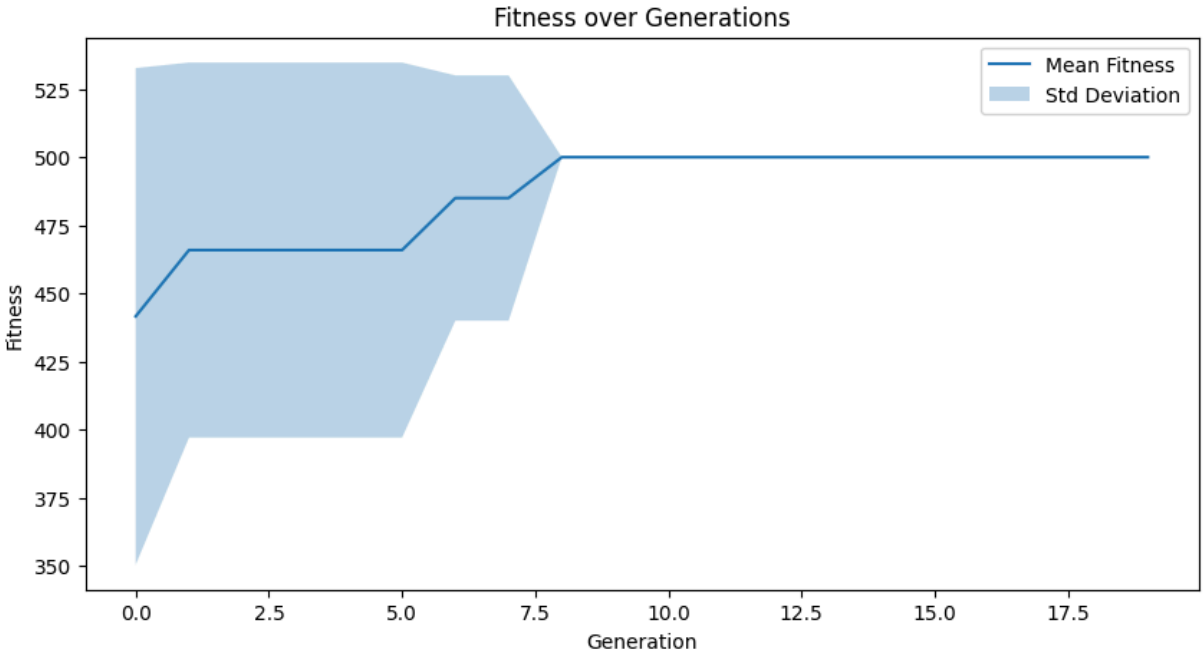
In [76]:
```
import numpy as np
import matplotlib.pyplot as plt
import gym

# Comments are not included in this section, as the code is the same as above.

# Define the CartPole evaluation function
def cartpole(x):
    inp, hid, out = 4, 8, 2
    input_weights, hidden_weights = x

    env = gym.make("CartPole-v1")
    observation, info = env.reset(seed=0)

    rew = 0
    step = 0
    done = False
    maxStep = 1000

    while not done and step < maxStep:
        hidden_layer_activation = np.dot(observation, input_weights)
        hidden_layer_output = np.tanh(hidden_layer_activation)
        output_layer_activation = np.dot(hidden_layer_output, hidden_weights)
        action = np.argmax(output_layer_activation)
        observation, reward, done, truncated, info = env.step(action)
        done = done or truncated
        step += 1
        rew += reward

    env.close()
    return np.minimum(maxStep, rew)

# Define the initialization function
def initialization(population_size):
    inp, hid, out = 4, 8, 2
    population = []
    for _ in range(population_size):
        input_weights = np.random.uniform(low=-0.1, high=0.1, size=(inp, hid))
        hidden_weights = np.random.uniform(low=-0.1, high=0.1, size=(hid, out))
        x = (input_weights, hidden_weights)
        population.append(x)
    return population

# Define the evaluation function
def evaluation(x, objective_function):

    fitness=[]

    for individual in x:
        fitness.append(objective_function(individual))
    #creates array of possible solutions
```

```
    return fitness

    # fitness = [objective_function(individual) for individual in x]
    # return fitness

# Define the crossover function
def crossover(parents, p_crossover):
    num_parents = len(parents)
    offspring = []
    for _ in range(num_parents // 2):
        parent1 = parents[np.random.randint(num_parents)]
        parent2 = parents[np.random.randint(num_parents)]

        input_weights1, hidden_weights1 = parent1
        input_weights2, hidden_weights2 = parent2

        child_input_weights = input_weights1.copy()
        child_hidden_weights = hidden_weights1.copy()

        if np.random.rand() < p_crossover:
            crossover_point = np.random.randint(1, input_weights1.size)
            child_input_weights = np.concatenate((input_weights1.flatten()[:crossov
                                                  input_weights2.flatten()[crossove

        if np.random.rand() < p_crossover:
            crossover_point = np.random.randint(1, hidden_weights1.size)
            child_hidden_weights = np.concatenate((hidden_weights1.flatten()[:cross
                                                   hidden_weights2.flatten()[crosso

        offspring.append((child_input_weights, child_hidden_weights))
    return offspring

# Define the mutation function
def mutation(x, mutation_rate):
    input_weights, hidden_weights = x
    if np.random.rand() < mutation_rate:
        input_weights += np.random.uniform(low=-0.1, high=0.1, size=input_weights.s
    if np.random.rand() < mutation_rate:
        hidden_weights += np.random.uniform(low=-0.1, high=0.1, size=hidden_weights
    return (input_weights, hidden_weights)

# Define the parent selection function
def parent_selection(x, f):
    num_parents = len(x) // 2
    sorted_parents = sorted(zip(f, x), key=lambda item: item[0], reverse=True)
    selected_parents = [item[1] for item in sorted_parents[:num_parents]]
    return selected_parents

# Define the survivor selection function
def survivor_selection(x, f, x_offspring, f_offspring):
    combined = list(zip(f + f_offspring, x + x_offspring))
    combined_sorted = sorted(combined, key=lambda item: item[0], reverse=True)
    f_new, x_new = zip(*combined_sorted[:len(x)])
    return list(x_new), list(f_new)

# Define the evolutionary algorithm function
```

```python
def ea(population_size, max_fit_evals, p_crossover, m_rate, objective_function):
    max_generations = max_fit_evals // population_size
    x = initialization(population_size)
    f = evaluation(x, objective_function)
    x_best = [x[np.argmax(f)]]
    f_best = [np.max(f)]

    for _ in range(max_generations - 1):
        parents = parent_selection(x, f)
        offspring = crossover(parents, p_crossover)
        offspring = [mutation(child, m_rate) for child in offspring]
        f_offspring = evaluation(offspring, objective_function)
        x, f = survivor_selection(x, f, offspring, f_offspring)

        idx = np.argmax(f)
        if f[idx] > f_best[-1]:
            x_best.append(x[idx])
            f_best.append(f[idx])
        else:
            x_best.append(x_best[-1])
            f_best.append(f_best[-1])

    return x_best, f_best

# Hyperparameters
kwargs = {
    "population_size": 100,
    "max_fit_evals": 2000,
    "p_crossover": 0.9,
    "m_rate": 0.05,
    "objective_function": cartpole,
}

# Run the algorithm 10 times and store the fitness values
num_runs = 10 # algorithm runs 10 times
all_fitness_values = [] #array to store fitness values over runtime

for run in range(num_runs):
    print(f"Run {run + 1}/{num_runs}")
    _, f_best = ea(**kwargs)
    all_fitness_values.append(f_best)

# Calculate mean and standard deviation of fitness values
all_fitness_values = np.array(all_fitness_values)
mean_fitness = np.mean(all_fitness_values, axis=0)
std_fitness = np.std(all_fitness_values, axis=0)

# Plot the results
generations = np.arange(len(mean_fitness))
plt.figure(figsize=(10, 5))
plt.plot(generations, mean_fitness, label='Mean Fitness')
plt.fill_between(generations, mean_fitness - std_fitness, mean_fitness + std_fitnes
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.title('Fitness over Generations')
```

```
plt.legend()
plt.show()
```

```
Run 1/10
Run 2/10
Run 3/10
Run 4/10
Run 5/10
Run 6/10
Run 7/10
Run 8/10
Run 9/10
Run 10/10
```



Question 4 (0-0.25-0.5 pt): Please comment on the average behavior of your algorithm. How did the average results and standard deviations look? Did your algorithm converge all the time to the best fitness?

Answer: The algorithm eventually converges to the best fitness value after multiple generations. It starts lower and eventually increases. The standard deviation fluctuates more at the beginning and then levels off by the 10th run. This all makes sense based on the definitions of averages and standard deviations. These results were not entirely consistent with those of my first trials. In the first part of the assignmnet, the fitness found is 1000 every time I run it, but in this part it only ever reaches 500. However, the fitness is still relatively high and extremely consistent. I think these results do align with those from my first trial, but I think the mistake lies in my graphing and not my algorithm, and it ultimately does converge to the best fitness possible.

In [ ]: