assignment_5_last!!

May 23, 2024

1 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

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

Imports

[ ]:
```
# Necessary libraries
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
# 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()
```

[2]:
```
%pip install gymnasium
import gymnasium as gym
```

```
Collecting gymnasium
  Downloading gymnasium-0.29.1-py3-none-any.whl (953 kB)
                             953.9/953.9
kB 6.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.21.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (1.25.2)
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium) (4.11.0)
Collecting farama-notifications>=0.0.1 (from gymnasium)
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Installing collected packages: farama-notifications, gymnasium
Successfully installed farama-notifications-0.0.4 gymnasium-0.29.1
```

Question 1 (0-0.25-0.5 pt): Following link provides more information about the CartPole environment 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:

Observation space in the CartPole environment consists of 4 real-valued numbers: - Cart Position, the position of the cart along the horizontal axis. Its values range from -4.8 to 4.8. - Cart Velocity is the velocity of the cart which values are between from -inf to +inf. - Pole Angle, vertically the angle of the pole, which ranges from -24 degrees to 24 degrees. - Pole Angular Velocity, the rating at the changing angle, ranges between min -inf to max inf.

Since each of these values can take any real number between their ranges the observation space is continuous. There are 4 observation so the dimension hereby is 4, the space represented as a 4-dimensional box.
Action Space: Action space contains two possible actions that make it discrete. These are binary choices for cart direction.

- 1. 0: Push cart to the left
- 2. 1: Push cart to the right

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:

ANN architecture

Input Layer

- Number of Neurons: 4

- Observation space of the CartPole environment has 4 dimensions as follows, cart position, cart velocity, pole angle, and pole angular velocity. The input layer, hereby, takes these continuous values as inputs.
  - Cart Position and Velocity provide the spatial and momentum-related information of ANN.
  - Pole Angle and Angular Velocity are responsible for predicting the stability of the pole, preventing it from falling.

Hidden Layers(s)

- Number of Neurons: 20
  - I added 20 neurons which support the fact that with more neurons the network can learn more complex patterns which helps the model in this case to choose the best weight options.
- Activation Function: Tanh

```
#Returns a vector which has values that transformed from the input
#observations through a weighted sum and action function
hidden_layer = np.tanh(np.dot(observation, w1))
```

- The output range of the Tanh is -1 to 1. That is why, the cases where the model has negative changes can be learned easily. It is zero-centered which means it handles the weights without bias in gradient updates. And this saves time compared to other activation functions like ReLU or Sigmoid. Additionally, Cartpole has symmetric behaviour system with its way of going left and right. In this scenario, negative and positive values might be handled easily by Tanh with the way of handling neg values as left and pos values as right. This way, the balance in both directions can be ensured.

Output layer

- Number of Neurons: 2.
  - There are two action spaces to be performed, either 0 or 1.
- Activation Funtion: Softmax.

```
# Convert output layer to PyTorch tensor for sotfmax
output_tensor = torch.from_numpy(output_layer).float()
probs = F.softmax(output_tensor, dim=0)
```

- Using softmax from PyTorch is super efficient. It is generally used on neural networks (also used in asgm 4). In this way, output action can be chosen by the highest probability distribution. Softmax transforms logits from the output layer into probability distributions over possible actions. These probabilities are set later on action indices (0 or 1 from action space).

Indirect Encoding of Network Weights

Hereby, indirect encoding is used. This is because of the broad exploration of solution space of indirect encoding. It can work with significant complexity too. Additionally, its nature is better for biological algorithm implementations, in this case, for EA. It's not one-to-one mapping from genotype to phenotype.

```
#reshaping the weights, genotype to phenotype
w1 = x[:inp * hid].reshape((inp, hid))  # transforms input layer to a hidden layer
w2 = x[inp * hid:].reshape((hid, out))  # maps the 20 neurons in the hidden layer to the
```

Action Selection Mechanism

```
#returns a vector as output layer, non-negative
output_layer = np.dot(hidden_layer, w2)

# Convert output layer to PyTorch tensor for sotfmax
output_tensor = torch.from_numpy(output_layer).float()
probs = F.softmax(output_tensor, dim=0) #finding the probabilities for output tensors

# Choose action based on probabilities
action = torch.multinomial(probs, 1).item()
```

Activation starts with returning the logits by output_layer which uses hidden_layer and w2 to calculate dot product for outputs. These logits are converted into tensors (from PyTorch) that are used to find the probability distributions with the help of multiclass Sigmoid: Softmax (From NN Lecture 8).

The probability of choosing the action corresponding to logit k using the softmax function is calculated as follows:

$$P(\text{action corresponding to logit } k) = \frac{e^{\text{logit}_k}}{\sum_j e^{\text{logit}_j}}$$

Where the numerator is the exponential of the logit for the action k. And denominator is the sum of the exponentials of all logits, ensuring the normalization of probabilities to sum up to 1.

With the help of probabilistic sampling by torch.multinomial, an action is set to either 0 or 1 as an index. Thus, the action returns an index that represents an action either left or right.

(- To illustrate this complicated system: Let's say we have a tensor like [0.7, 0.3]. The selection of each action is based on the probability in the distribution. If Action 0 has a probability of 30% and Action 1 has 70%, in this case, the selected frequency of 1 would be higher. However, there would also always be a 30% chance of Action 0 being selected.)

This approach ensures algorithm to learn complex patterns by exploring different actions.

(
- Source_1: https://towardsdatascience.com/creating-deep-neural-networks-from-scratch-an-introduction-to-reinforcement-learning-part-i-549ef7b149d2

- Source_2: https://pythonprogramming.net/openai-cartpole-neural-network-example-machine-learning-tutorial/
- Source_3: https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html
- Source_4: https://pytorch.org/docs/stable/generated/torch.multinomial.html

)

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.

[3]:
```
##################################
# 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␣
 ↪explained
# 3.8 pts if the code works but some functions are incorrect but it is␣
 ↪explained well
# 4.2 pts if the code works very well aligned with the task without any␣
 ↪mistakes, but it is badly explained
# 5 pts if the code works very well aligned with the task without any mistakes,␣
 ↪and it is well explained
#####################################################################
import gymnasium as gym
import os
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F


# Artificial Neural Network parameters (weights)
# See here: https://www.gymlibrary.dev/environments/classic_control/cart_pole/␣
 ↪for input and output space
# PLEASE SPECIFY BELOW
inp = 4 # Number of input neurons
hid = 20 # Number of hidden neurons
out = 2 # Nubmer of output neurons
####################

video_folder = "content/video/11"
os.makedirs(video_folder, exist_ok=True)

#Open AI gym environment
env = gym.make("CartPole-v1")
env._max_episode_steps = 1000

# CartPole evaluation function
def cartpole(x):

    ############
    # PLEASE FILL IN
    # Hint: x is an individual in evolutionary algorithms and needs to map to␣
 ↪the connection weights of ANNs
```

```
   #reshaping the weights
   #indirect encoding
   w1 = x[:inp * hid].reshape((inp, hid)) #transforms input layer to a hidden layer
   w2 = x[inp * hid:].reshape((hid, out)) #maps the 20 neurons in the hidden layer to the 2 output neurons
   #########

   # 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:

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

     #Returns a vector which has values that transformed from the input
     #observations through a weighted sum and action function
     hidden_layer = np.tanh(np.dot(observation, w1))

     #returns a vector as output layer, non-negative
     output_layer = np.dot(hidden_layer, w2)


     # Convert output layer to PyTorch tensor for sotfmax
     output_tensor = torch.from_numpy(output_layer).float()
     probs = F.softmax(output_tensor, dim=0) #finding the probabilities for output tensors

     # Choose action based on probabilities
     #Returns an index based on probability distribution --> 1 or 0.
     action = torch.multinomial(probs, 1).item()
     #print(action)


     # action should be provided based on the output of the artifial neural network
     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
 ↪< 1000, this means that pole fell)



# 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
 ↪see the video here:content/video/cartpole
    env = gym.wrappers.RecordVideo(env=tmp_env, video_folder="content/video/
 ↪cartpole", name_prefix="cartpole")

    ############
    # PLEASE FILL IN
    #reshaping the weights
    w1 = x[:inp * hid].reshape((inp, hid)) #transforms input layer to a hidden
 ↪layer
    w2 = x[inp * hid:].reshape((hid, out)) #maps the 20 neurons in the hidden
 ↪layer to the 2 output neurons
    #########
    #########

    # 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


      ###################
      # Hint: Provide input to ANN and find the output to be the action

      #returns a vector which has values that transformed from the input
      #observations throigh a weighted sum and action function
      hidden_layer = np.tanh(np.dot(observation, w1))

      #returns a vector as output layer, non-negative
      output_layer = np.dot(hidden_layer, w2)

      # Convert output layer to PyTorch tensor for sotfmax
```

```
      output_tensor = torch.from_numpy(output_layer).float()
      probs = F.softmax(output_tensor, dim=0) #finding the probabilities for␣
↪output tensors

      # Choose action based on probabilities
      action = torch.multinomial(probs, 1).item()



      # action should be provided based on the output of the artifial neural␣
↪network
      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␣
↪< 1000, this means that pole fell)



"""
# 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 connection weights of ANNs

    #########

    # 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␣
 ↪network
      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␣
 ↪< 1000, this means that pole fell)
"""

# Necessery functions for EA part


# Function to initialize a population of weights
# each individual in the population is represented by a vector of weights␣
 ↪(genes).
def initialize_population(population_size, gene_length):

    #print(np.random.randn(population_size, gene_length) * 0.1)
    return np.random.randn(population_size, gene_length) * 0.1  # Scale down␣
 ↪the weights for computaiton efficiency


# Tournament selection for parent seelctio
# from my parent_selection function of asgm 2
def tournament_selection(population, scores, k=3):
    selected_indices = np.random.randint(len(population), size=k)
    best_index = selected_indices[np.argmax(scores[selected_indices])]
    return population[best_index]

# Crossover
# from my asgm 2
# single-point crossover
def crossover(parent1, parent2, p_crossover):
    if np.random.rand() < p_crossover:
        crossover_point = np.random.randint(len(parent1))
        child1 = np.concatenate([parent1[:crossover_point],␣
 ↪parent2[crossover_point:]])
        child2 = np.concatenate([parent2[:crossover_point],␣
 ↪parent1[crossover_point:]])
        return child1, child2
    else:
```

```
        return parent1.copy(), parent2.copy()

# Mutation
#According to the mutation rate setting Gaussian noise (mean=0, std=0.1) to the␣
 ↪genes
# source: https://wildart.github.io/Evolutionary.jl/dev/mutation/ AND Lecture 9
def mutate(individual, mutation_rate):
    mutation_mask = np.random.rand(len(individual)) < mutation_rate
    individual[mutation_mask] += np.random.randn(np.sum(mutation_mask)) * 0.1 ␣
 ↪#random variation
    return individual

# Implement your Evolutionary Algorithm to find the ANN weigths that can␣
 ↪balance the CartPole
# Feel free to add any functions, such as initialization, crossover, etc.. to␣
 ↪make it work!
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
):
    ####################
    # Hint: your implementation of your evolutionary algorithm
    # You may use the code you previously implemented during the course


    #Indirect encoding
    #Population initialization
    gene_length = inp * hid + hid * out
    population = initialize_population(population_size, gene_length)

    f_best = -np.inf
    x_best = None

    for generation in range(max_fit_evals // population_size):
        fitness = np.array([objective_function(ind) for ind in population])␣
 ↪#fitness scores calculated by cartpole obj func
        new_population = []

        ##generating new population
        for _ in range(population_size // 2):
            # Selecting two parents based on their fitness scores
            parent1 = tournament_selection(population, fitness)
            parent2 = tournament_selection(population, fitness)
```

```
            # using selected parents to generate new offspring, childeredn
            child1, child2 = crossover(parent1, parent2, p_crossover)

            # Introduce variation by applying mutation
            child1 = mutate(child1, m_rate)
            child2 = mutate(child2, m_rate)

            # Adding mutations into the population
            new_population.extend([child1, child2])

        # Replacing old population with the new population
        population = np.array(new_population[:population_size])

        # Update max fitness
        current_max_fitness = max(fitness)
        if current_max_fitness > f_best:  #See if the new fitness higher
            f_best = current_max_fitness
            best_index = np.argmax(fitness)  # Find the index of the individual␣
↪with the best fitness

            # Updating the best solution found so far
            if best_index < len(population):
                x_best = population[best_index]

        #print(f"Generation {generation}: Best fitness {f_best}")

    return x_best, f_best  # return the best solution (ANN weights) and the␣
↪fitness in each generation
```

Check Your Implementation: Running The Evolutionary Algorithm Run the cell below, if you implemented everything correctly, you should see the algorithm running. Furthermore,

[7]:
```
# Dummy parameters, please add or remove based on your implementation
kwargs = {
    "population_size": 15,
    "max_fit_evals": 1000,  # maximum number of fitness evaluations
    "p_crossover": 0.9,  # crossover probability
    "m_rate": 0.5,  # mutation rate
    "objective_function": cartpole,
}
# 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)
print("Best fitnes found:",f_best)


# Evaluate your ANN weights again and record the video
if f_best >= 1000:
  cartpole_record_video(x_best)
  #or cartpole_visualize(x_best)
else:
  print("The best fitness 1000 was not found, try again!!")
```

```
Best ANN parameters found: [ 0.2564232  -0.49448532  0.06220058 -0.32346884
-0.76017674 -0.87221554
 -0.09160606 -0.4560121   0.45411484  0.19001373 -0.03642772 -0.77902469
 -0.22851546 -0.26005281 -0.09567809  1.02661862 -0.63452938  0.22204301
  0.07372324  0.70098077  0.4699686   0.32472512 -0.89509859 -0.38813352
  0.09435325 -0.21633859 -0.1643765   0.30547446  0.15839306  0.40503507
  0.28757172  0.68888878 -0.67831837  0.49569406  0.7222071  -0.22387452
  0.03000733 -0.02459962 -0.63215751 -0.07356056  0.52824592 -0.51653352
  0.2556489  -0.24324338  0.05874264 -0.30266121 -0.74496652  0.98739432
  0.70042536 -0.67582193  0.19495089 -0.07315118 -0.455937    0.32092017
 -0.03907281 -0.50782612  0.85793896 -0.40796214 -0.57067778  0.36018848
  0.4820514   0.19865982 -0.13748011 -0.1654347  -0.59949699 -0.02515827
 -0.76035491 -0.57177491 -0.31285634  0.3338124  -0.17512858  0.40559971
  1.02992821  1.43133137 -1.05310446  0.36998084 -0.35331581 -0.62174771
 -0.13844221 -0.0834481  -0.739623    0.55246785  0.24104747 -0.05478895
  0.72371928  0.85721697  1.07740325  0.20699196 -0.08438455 -0.59096922
  0.07054799 -0.93607154 -0.25377737 -0.97241581 -0.42877003 -0.95019053
 -0.23930696  0.37022395  0.09130036  0.26173608 -0.16137382  0.24823562
  0.23255591  0.61462093 -0.77184145 -0.05965752 -0.90786901  1.03478753
  0.04360311 -1.05258787  0.40713824 -1.40358347 -0.11541086  0.19428478
  0.63996427 -1.35913691  0.67859424 -0.71542181 -0.3052159   0.51464121]
Best fitnes found: 1000.0
Moviepy - Building video /content/content/video/cartpole/cartpole-episode-0.mp4.
Moviepy - Writing video /content/content/video/cartpole/cartpole-episode-0.mp4



Moviepy - Done !
Moviepy - video ready /content/content/video/cartpole/cartpole-episode-0.mp4


Moviepy - Building video /content/content/video/cartpole/cartpole-episode-0.mp4.
Moviepy - Writing video /content/content/video/cartpole/cartpole-episode-0.mp4
```

```
Moviepy - Done !
Moviepy - video ready /content/content/video/cartpole/cartpole-episode-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 achieve a fitness score of 1000 to determine ANN weights. However, since EAs are stochastic algorithms, unfortunately, I did not receive a fitness score of 1000 in every run. The reason of this might be the random weight initialization and genetic operations like mutation and crossover. In my case, I recorded over 10 videos with 1000 fitness, and in none of them did the pole fall. In the last final recorded video, It moves left and right without stopping. There is a balanced movement from left to right, and right to left during the video. It doesn't bounce or get stuck. It behaves as expected from the task. Moves horizontally with the given actions 1 and 0.

I believe this is because of the model's performance. EA in this case improved over generations and found the optimal weights for the cartpole. The evaluator of the fitness in this case was the cartpole function which served as the objective function. It provided immediate feedback for the selected weights and helped the model to improve itself during EA performance to select the best weights. Additionally, the objective function was designed to keep the pole balanced. The activation functions in this case were specifically selected for the cartpole environment (as I stated in question 2).

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.

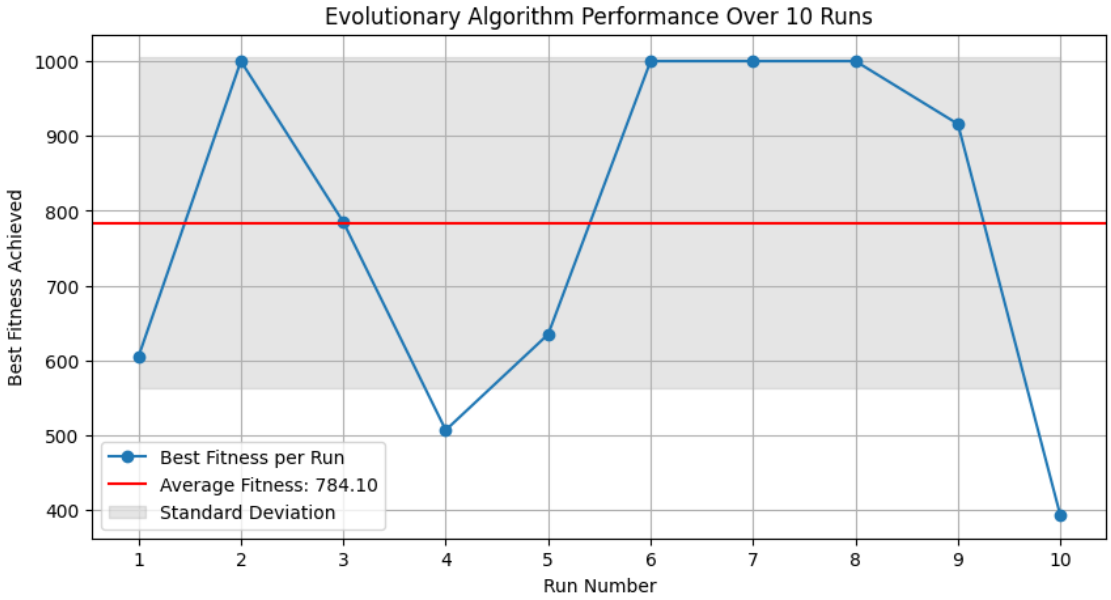
Task 2 (0-1.5-3 pt): Please run your algorithm for at least 10 times and plot the average results and standard deviations. Below, you may add as many cells as you need for this implementation and plot functions. You may use previous code you have developed/used during the course.

[62]:
```python
import matplotlib.pyplot as plt

def run_algorithm_10_times(runs, kwargs):
    """
    Simple function returns best fitness each run.
    Runs the algorithm 10 times
    Parameters: runs, kwargs

    """
    best_fitnesses = []
    for _ in range(runs):
        _, f_best = ea(**kwargs)
```

```
        best_fitnesses.append(f_best)
    return best_fitnesses

best_fitnesses = run_algorithm_10_times(runs=10, kwargs=kwargs)
```

[64]:
```
## Source: https://learnpython.com/blog/average-in-matplotlib/
## Source: https://realpython.com/visualizing-python-plt-scatter/


def plot_results(best_fitnesses):
    plt.figure(figsize=(10, 5))
    plt.title("Evolutionary Algorithm Performance Over 10 Runs")

    #x-as y-as
    plt.xlabel("Run Number")
    plt.ylabel("Best Fitness Achieved")

    ## Ploting the graphs
    plt.plot(range(1, len(best_fitnesses) + 1), best_fitnesses, 'o-',␣
 ↪label='Best Fitness per Run')
    plt.axhline(y=np.mean(best_fitnesses), color='r', linestyle='-',␣
 ↪label=f'Average Fitness: {np.mean(best_fitnesses):.2f}')

    # np.std responsible for Population SD
    plt.fill_between(range(1, len(best_fitnesses) + 1),
                     np.mean(best_fitnesses) - np.std(best_fitnesses),
                     np.mean(best_fitnesses) + np.std(best_fitnesses),
                     color='gray', alpha=0.2, label='Standard Deviation')

    plt.xticks(range(1, len(best_fitnesses) + 1))  # Add run numbers on x-axis
    plt.legend()
    plt.grid(True)
    plt.show()

def results_table(best_fitnesses):
    """
    Displays a result table, which all the values are clearly stated.

    """
    results_df = pd.DataFrame({
        'Run Number': range(1, len(best_fitnesses) + 1),
        'Best Fitness': best_fitnesses
    })
    results_df['Mean'] = np.mean(best_fitnesses)
    results_df['Standard Deviation'] = np.std(best_fitnesses)
    print(results_df)
```

```
plot_results(best_fitnesses)
results_table(best_fitnesses)
```



	Run Number	Best Fitness	Mean	Standard Deviation
0	1	605.0	784.1	220.884789
1	2	1000.0	784.1	220.884789
2	3	785.0	784.1	220.884789
3	4	507.0	784.1	220.884789
4	5	635.0	784.1	220.884789
5	6	1000.0	784.1	220.884789
6	7	1000.0	784.1	220.884789
7	8	1000.0	784.1	220.884789
8	9	916.0	784.1	220.884789
9	10	393.0	784.1	220.884789

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 average result of the algorithm is around 784.1, which can be considered as high, taking into account of 10 runs.

The formula used in the code for SD, in this case population SD (ddof=0) is:

`np.std`

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

- μ: mean of the data
- x_i: individual fitness scores obtained in each run
- N: total number of runs

Hereby, with this formula, the achieved SD is around 220.88. This can be accepted as normal considering the obtained results of the fitness scores in each run. Particularly, the scores of the 5th and 10th runs are 635.0, and 393.0, respectively. These values are way too low compared to other runs which contribute to the observed variability along with its significant impact on SD. This is because of the stochastic nature of the EAs that in each run the fitness scores change a lot due to the genetic variations and selection processes within the algorithm. So no my algorithm did not converge all the time the best fitness score.

(Source for sdt: https://numpy.org/doc/stable/reference/generated/numpy.std.html and https://www.sharpsightlabs.com/blog/numpy-standard-deviation/)