Elective Course
Course Code: CS4103
Autumn 2025-26

Lecture #23

Artificial Intelligence for Data Science

Week-7:
Exploring genetic algorithm (GA) using PyGAD
Introduction to Knowledge Representation and Logic [Part-II]

Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computational and Data Sciences
Indian Institute of Science Education and Research Kolkata, India 741246

A Few Python Libraries for AI

2

easyAI
PyGAD
Pytholog
pgmpy
scikit learn
NetworkX Network Analysis in Python
matplotlib
NumPy
pandas

PyGAD

- PyGAD is an intuitive library for optimization using the genetic algorithm.
- The modules included in PyGAD are:
  1) pygad.pygad
  2) pygad.utils
  3) pygad.helper
  4) pygad.visualize
  5) pygad.nn
  6) pygad.gann
  7) pygad.cnn
  8) pygad.gacnn
  9) pygad.kerasga
  10) pygad.torchga

Using PyGAD

- There are 3 core steps to use PyGAD:
  – 1) Build the fitness function.
  – 2) Instantiate the pygad.GA class with the appropriate configuration parameters.
  – 3) Call the run() method to start the evolution.

Using PyGAD: Example
(best weight estimation)

$$y = w_1x_1 + w_2x_2 + w_3x_3$$
$$where\ \ y = 100,\ x_1 = 2,\ x_2 = 4,\ x_3 = 6$$

```
import pygad
import numpy

function_inputs = [2,4,6] # Function inputs.
desired_output = 100 # Function output.

def fitness_func(ga_instance, solution, solution_idx):
    output = numpy.sum(solution*function_inputs)
    fitness = 1.0 / numpy.abs(output - desired_output)
    return fitness

fitness_function = fitness_func

num_generations = 100 # Number of generations.
num_parents_mating = 7 # Number of solutions to be selected as parents in the mating pool.
sol_per_pop = 50 # Number of solutions in the population.
num_genes = len(function_inputs)
```


Using PyGAD: Example
(best weight estimation)

```
last_fitness = 0
def callback_generation(ga_instance):
    global last_fitness
    print(f"Generation = {ga_instance.generations_completed}")
    print(f"Fitness    = {ga_instance.best_solution()[1]}")
    print(f"Change     = {ga_instance.best_solution()[1] - last_fitness}")
    last_fitness = ga_instance.best_solution()[1]

# Creating an instance of the GA class inside the ga module. Some parameters are
initialized within the constructor.
ga_instance = pygad.GA(num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       on_generation=callback_generation)
# Running the GA to optimize the parameters of the function.
ga_instance.run()
```

Using PyGAD: Example
(best weight estimation)

```
# After the generations complete, some plots are showed that summarize the how
the outputs/fitenss values evolve over generations.
ga_instance.plot_fitness()


# Returning the details of the best solution.
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print(f"Parameters of the best solution : {solution}")
print(f"Fitness value of the best solution = {solution_fitness}")
print(f"Index of the best solution : {solution_idx}")

prediction = numpy.sum(numpy.array(function_inputs)*solution)
print(f"Predicted output based on the best solution : {prediction}")

if ga_instance.best_solution_generation != -1:
    print(f"Best fitness value reached after
{ga_instance.best_solution_generation} generations.")
```

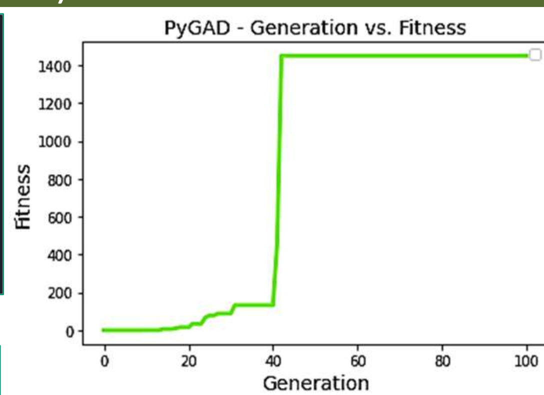

Using PyGAD: Example
(best weight estimation)

```
Generation = 1
Fitness    = 0.016693849903584405
Change     = 0.016693849903584405
Generation = 2
Fitness    = 0.018279913896577488
Change     = 0.0015860639929930823
Generation = 3
Fitness    = 0.01895767419492364
Change     = 0.0006777602983461528
Generation = 4
Fitness    = 0.020417782656941927
Change     = 0.0014601084620182864
Generation = 5
Fitness    = 0.025766762542942575
Change     = 0.005348979886000648
Generation = 6
Fitness    = 0.03025527917430055
Change     = 0.004488516631357974
Generation = 7
Fitness    = 0.03115319280992052
Change     = 0.00089791363561997
...
...
```

```
Generation = 41
Fitness    = 460.2805922285196
Change     = 328.0223091723938
Generation = 42
Fitness    = 1449.0388143680036
Change     = 988.758222139484
Generation = 43
Fitness    = 1449.0388143680036
Change     = 0.0
Generation = 44
Fitness    = 1449.0388143680036
Change     = 0.0
```

...

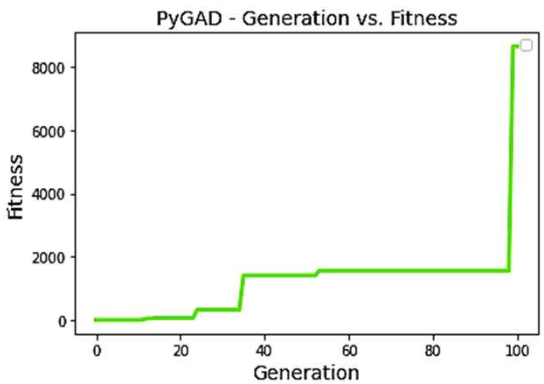
```
Generation = 98
Fitness    = 1449.0388143680036
Change     = 0.0
Generation = 99
Fitness    = 1449
Change     = 0.0
Generation = 100
Fitness    = 1449
Change     = 0.0
```

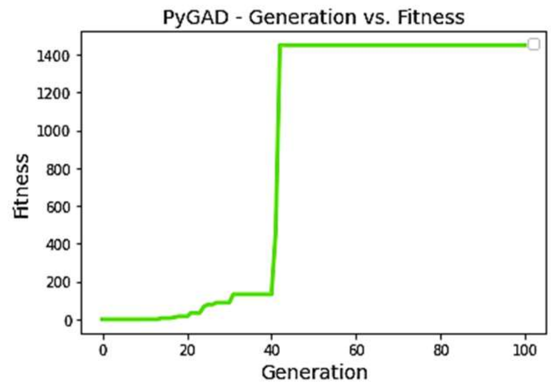



```
Parameters of the best solution : [2.27844223 9.21225828
9.76579543]
Fitness value of the best solution = 1449.0388143680036
Index of the best solution : 0
Predicted output based on the best solution :
100.00069011263886
Best fitness value reached after 42 generations.
```

Using PyGAD: Example
(best weight estimation)

```
Parameters of the best solution : [6.22756165 8.29135122
9.06642279]
Fitness value of the best solution = 52.45243293178735
Index of the best solution : 45
Predicted output based on the best solution :
100.01906489259136
Best fitness value reached after 99 generations.
```






```
Parameters of the best solution : [2.27844223 9.21225828
9.76579543]
Fitness value of the best solution = 1449.0388143680036
Index of the best solution : 0
Predicted output based on the best solution :
100.00069011263886
Best fitness value reached after 42 generations.
```


Using PyGAD: Example
(4-Queen Problem Solving)

```
N = 4 # For 4-Queens
def fitness_func(ga_instance, solution_vector, solution_idx):
    if solution_vector.ndim == 2:
        solution = solution_vector
    else:
        solution = np.zeros(shape=(4, 4))
        row_idx = 0
    for col_idx in solution_vector:
        solution[row_idx, int(col_idx)] = 1
        row_idx = row_idx + 1

    total_num_attacks = compute_tot_attack(solution)??

    if total_num_attacks == 0:
        total_num_attacks = float("inf")
    else:
        total_num_attacks = 1.0/total_num_attacks
    return total_num_attacks
```

Home Assignment

Computing total attack, given 2D array (solution) as (e.g.):

		1	
	1		
1			
		1	

Using PyGAD: Example
(4-Queen Problem Solving)

11

```
ga_instance = pygad.GA(num_generations=200,
                       num_parents_mating=10,
                       fitness_func=fitness_func,
                       sol_per_pop=50,
                       num_genes=N,
                       gene_type=int,
                       gene_space=range(N),
                       crossover_type="single_point",
                       mutation_type="random",
                       mutation_percent_genes=10)

ga_instance.run()

best_solution, best_solution_fitness, best_solution_idx =
ga_instance.best_solution()
print(f"Best solution: {best_solution}")
print(f"Fitness of the best solution: {best_solution_fitness}")
```

```
Best solution: [2 0 3 1]
Fitness of the best solution: inf
```

		Q	
Q			
			Q
	Q		

Home Assignment
Given the solution, implement the visualization part

(Both assignments together in a single code to finally solve 4-Queen problem)

Submission:
Submit by **03-OCT-2025** (through email with subject "CS4103: Assignment on solving 4-Queen using PyGAD")
Please write the code yourself. DO NOT use AI tools to develop the code.

12

Knowledge Representation and Logic
[contd.]

Logic

13

A logic is a triple $\langle \mathcal{L}, \mathcal{S}, \mathcal{R} \rangle$ where

- $\mathcal{L}$, the logic's language, is a class of sentences described by a formal grammar
- $\mathcal{S}$, the logic's semantics is a formal specification of how to assign *meaning* in the "real world" to the elements of $\mathcal{L}$
- $\mathcal{R}$, the logic's inference system, is a set of formal derivation *rules* over $\mathcal{L}$

There are several logics: propositional, first-order, higher-order, modal, temporal, intuitionistic, linear, equational, non-monotonic, fuzzy, ...

We will concentrate on propositional logic and first-order logic

Propositional Logic

14

Each sentence is made of

- propositional variables $(A, B, \ldots, P, Q, \ldots)$
- logical constants $(\mathbf{True}, \mathbf{False})$
- logical connectives $(\wedge, \vee, \Rightarrow, \ldots)$

Every propositional variable stands for a basic fact

- Examples: I'm hungry, Apples are red, Sky is blue

Propositional Logic

15

The Language

- Each propositional variable ($A, B, \ldots, P, Q, \ldots$) is a sentence
- Each logical constant ($\mathbf{True}, \mathbf{False}$) is a sentence
- If φ and ψ are sentences, all of the following are also sentences

$$(\varphi) \qquad \neg\varphi \qquad \varphi \wedge \psi \qquad \varphi \vee \psi \qquad \varphi \Rightarrow \psi \qquad \varphi \Leftrightarrow \psi$$

- Nothing else is a sentence

The Language of Propositional Logic

16

More formally, it is the language generated by the following grammar

Grammar Rules:

$$\begin{array}{lcl}
Sentence & ::= & AtomicS \mid ComplexS \\
AtomicS & ::= & \mathbf{True} \mid \mathbf{False} \mid A \mid B \mid \ldots \mid P \mid Q \mid \ldots \\
ComplexS & ::= & (Sentence) \mid Sentence\ Connective\ Sentence \mid \neg Sentence \\
Connective & ::= & \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow
\end{array}$$

Semantics of Propositional Logic

17

The meaning of **True** is always *true*
The meaning of **False** is always *false*

The meaning of the other sentences depends on the meaning of the propositional variables

- Base cases: truth tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

- Non-base Cases: given by reduction to the base cases
  Ex: the meaning of $(P \vee Q) \wedge R$ is the same as the meaning of $A \wedge R$ where A has the same meaning as $P \vee Q$

Semantics of Propositional Logic

18

An assignment of Boolean values to the propositional variables of a sentence is an interpretation of the sentence

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
false	*false*	*false*	*false*	*true*
false	*true*	*true*	*false*	*true*
true	*false*	*true*	*true*	*true*
true	*true*	*true*	*false*	*true*

Interpretations: $\{P \mapsto \mathit{false}, H \mapsto \mathit{false}\}, \{P \mapsto \mathit{false}, H \mapsto \mathit{true}\}, \ldots$

An interpretation is a model of a sentence φ if it makes the sentence true

Semantics of Propositional Logic

19

The meaning of a sentence in general depends on its interpretation
Some sentences, however, have always the same meaning

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
false	*false*	*false*	*false*	*true*
false	*true*	*true*	*false*	*true*
true	*false*	*true*	*true*	*true*
true	*true*	*true*	*false*	*true*

A sentence is

- satisfiable if it is true in some interpretation
- unsatisfiable if it is true in no interpretation
- valid if it is true in every possible interpretation
- invalid if it is false in some possible interpretation

Entailment in Propositional Logic

20

Given

- a set Γ of sentences and
- a sentence φ,

we write

$$\Gamma \models \varphi$$

iff every interpretation that makes all sentences in Γ true makes φ also true

$\Gamma \models \varphi$ is read as "Γ entails φ" or "φ logically follows from Γ"

Entailment in Propositional Logic

21

Examples

$$\begin{array}{lcl} \{A, A \Rightarrow B\} & \models & B \\ \{A\} & \models & A \vee B \\ \{A, B\} & \models & A \wedge B \\ \{\} & \models & A \vee \neg A \\ \{A\} & \not\models & A \wedge B \\ \{A \vee \neg A\} & \not\models & A \end{array}$$

	A	B	$A \Rightarrow B$	$A \vee B$	$A \wedge B$	$A \vee \neg A$
1.	*false*	*false*	*true*	*false*	*false*	*true*
2.	*false*	*true*	*true*	*true*	*false*	*true*
3.	*true*	*false*	*false*	*true*	*false*	*true*
4.	*true*	*true*	*true*	*true*	*true*	*true*

Properties of Entailment

22

- $\Gamma \models \varphi$, for all $\varphi \in \Gamma$ (inclusion property of PL)
- if $\Gamma \models \varphi$, then $\Gamma' \models \varphi$ for all $\Gamma' \supseteq \Gamma$ (monotonicity of PL)
- φ is valid iff $\{\,\} \models \varphi$ (also written as $\models \varphi$)
- φ is unsatisfiable iff $\varphi \models \mathbf{False}$
- $\Gamma \models \varphi$ iff the set $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable

Logical Equivalence

23

Two sentences φ_1 and φ_2 are logically equivalent, written

$$\varphi_1 \equiv \varphi_2$$

if $\varphi_1 \models \varphi_2$ and $\varphi_2 \models \varphi_1$

Note:

- $\varphi_1 \equiv \varphi_2$ if and only if every interpretation assigns the same Boolean value to φ_1 and φ_2

Properties of Logical Connectives

24

- $\wedge$ and $\vee$ are commutative

$$\begin{aligned} \varphi_1 \wedge \varphi_2 &\equiv \varphi_2 \wedge \varphi_1 \\ \varphi_1 \vee \varphi_2 &\equiv \varphi_2 \vee \varphi_1 \end{aligned}$$

- $\wedge$ and $\vee$ are *associative*

$$\begin{aligned} \varphi_1 \wedge (\varphi_2 \wedge \varphi_3) &\equiv (\varphi_1 \wedge \varphi_2) \wedge \varphi_3 \\ \varphi_1 \vee (\varphi_2 \vee \varphi_3) &\equiv (\varphi_1 \vee \varphi_2) \vee \varphi_3 \end{aligned}$$

- $\wedge$ and $\vee$ are mutually *distributive*

$$\begin{aligned} \varphi_1 \wedge (\varphi_2 \vee \varphi_3) &\equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3) \\ \varphi_1 \vee (\varphi_2 \wedge \varphi_3) &\equiv (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \end{aligned}$$

- $\wedge$ and $\vee$ are related by $\neg$ (DeMorgan's Laws)

$$\begin{aligned} \neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) &\equiv \neg\varphi_1 \wedge \neg\varphi_2 \end{aligned}$$

Properties of Logical Connectives

25

$\wedge$, $\Rightarrow$, and $\Leftrightarrow$ are actually redundant:

$$\begin{aligned}
\varphi_1 \wedge \varphi_2 &\equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 &\equiv \neg\varphi_1 \vee \varphi_2 \\
\varphi_1 \Leftrightarrow \varphi_2 &\equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)
\end{aligned}$$

We keep them all mainly for convenience

Rule-Based Inference in PL

26

An inference system in Propositional Logic can also be specified as a set $\mathcal{R}$ of inference (or derivation) rules

- Each rule is just a *pattern* premises/conclusion
- A rule applies to Γ and derives φ if
  - some of the sentences in Γ match with the premises of the rule and
  - φ matches with the conclusion
- A rule is sound if the set of its premises entails its conclusion

Some Inference Rules

27

- And-Introduction

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

- And-Elimination

$$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{\alpha \wedge \beta}{\beta}$$

- Or-Introduction

$$\frac{\alpha}{\alpha \vee \beta} \qquad \frac{\alpha}{\beta \vee \alpha}$$

Some Inference Rules (cont'd)

28

- Implication-Elimination (aka Modus Ponens)

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$$

- Unit Resolution

$$\frac{\alpha \vee \beta \quad \neg\beta}{\alpha}$$

- Resolution

$$\frac{\alpha \vee \beta \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$ or, equivalently,

$$\frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Some Inference Rules (cont'd)

29

- Double-Negation-Elimination

$$\frac{\neg\neg\alpha}{\alpha}$$

- **False**-Introduction

$$\frac{\alpha \quad \neg\alpha}{\textbf{False}}$$

- **False**-Elimination

$$\frac{\textbf{False}}{\beta}$$

30

Questions?