Tips for Implementing Genetic Programming using Python

Hengzhe Zhang

Supervisor: Mengjie Zhang, Bing Xue, Qi Chen, Wolfgang Banzhaf (MSU)

Victoria University of Wellington

08/11/2023

Table of Contents

Python Basics

Defining a Function

There are two ways to define a Python function:

```
def weather(date):
        return Google(date)
```

And here is another way:

```
weather=lambda date: Google(date)
```

A List of Numbers

- `len(ind1)` represents measuring the length of an object. For a GP tree, it has been overridden by a function to calculate the number of nodes in a GP tree.
- `range(a, b)` iterates from a to b, not including b.

```
list(range(0, len(ind1)))
```

Please be aware of the comma:

- `(-1.0)`: This is a number.
- `(-1.0,)`: This is an immutable list (cannot insert/delete elements) with only one element.

GP for Symbolic Regression

Let's start with a simple case. We only need two lines of code to get the fitness value of a GP tree.

- `gp.compile` means to compile a GP tree using the Python interpreter.
- `psets` means a map of function and terminal names to real functions and terminals.

```
# Define the symbolic regression problem
def evalSymbReg(individual, pset):
        # Compile the GP tree into a function
        func = gp.compile(expr=individual, pset=pset)

        # Calculate Mean Square Error (MSE)
        mse = ((func(x) - x**2)**2 for x in range(-10, 10))
        return (math.fsum(mse),)
```

GP FOR TSP

- GP is usually used for evolving heuristic rules in TSP.
- The whole process is very complex. A complete example is available on the GitHub repository (last page slide).

```
current_city = route[-1]
next_city = select_next_city(current_city,
unvisited_cities, heuristic, distance_matrix)
```

Fitness Initialization

- Next, initialize the fitness vector. For a single objective, there is only one value, but it is still in vector form.
- The convention of DEAP is for solving maximization problems, thus we need to use a negative weight.

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree,
fitness=creator.FitnessMin)
```

Manually Compile GP Trees

Manually Compile GP Trees

- Sometimes, you may find the default Python compiler is slow. In this case, manually compiling GP trees is a good choice.
- This function can be directly invoked using the Python package "EvolutionaryForest." Only one line!

```
quick_evaluate(individual,pset,x)
```

Multi-tree GP

There are many ways to define multi-tree GP. Although I recommend you define a Python class, the simplest way is to use a list. Only one line!
From:

```
creator.create("Individual", gp.PrimitiveTree,
fitness=creator.FitnessMin)
```

To:

```
creator.create("Individual", list, fitness=creator.FitnessMax)
```

Multi-objective GP

Multi-objective Fitness

Want multi-objective GP? It's very easy; just define two weights.

```
creator.create("FitnessMulti", base.Fitness,
weights=(-1.0, -1.0))
```

Multi-objective Fitness

Don’t forget to return multiple fitness values.

```
        size = len(individual)
        return math.fsum(mse), size
```

Function Set / Terminal Set

Function Set / Terminal Set

Let's go back to a single tree after defining the fitness function.
Another important thing is to define operators and terminals.

- For terminals, if you have only one terminal, just set the arity as 1.
- For operators, any function can be passed.

```
pset = gp.PrimitiveSet("MAIN", arity=1)
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)
pset.addPrimitive(operator.neg, 1)
pset.addEphemeralConstant("rand101",
        lambda: random.randint(-1, 1))
```

Not limited to basic operators, you can define whatever you want! For example, connecting to the internet to get today's weather.

```
def weather(date):
        return Google(date)
pset.addPrimitive(weather, 1)
```

Numpy Speedup

Numpy Speedup

Pure Python is slow. When writing GP code in Python, please always make sure you are using NumPy functions.

```
pset = gp.PrimitiveSet("MAIN", arity=1)
pset.addPrimitive(np.add, 2)
pset.addPrimitive(np.subtract, 2)
pset.addPrimitive(np.multiply, 2)
pset.addPrimitive(np.negative, 1)
```

GPU Speedup

GPU Speedup

- When you have a large amount of data, you can even use GPU to calculate fitness values. Just replace basic functions with Torch functions.
- However, when using Torch functions, please make sure to load data to the GPU using PyTorch. Otherwise, the GPU cannot access your data.

```
pset = gp.PrimitiveSet("MAIN", arity=1)
pset.addPrimitive(torch.add, 2)
pset.addPrimitive(torch.sub, 2)
pset.addPrimitive(torch.mul, 2)
pset.addPrimitive(torch.neg, 1)
```

Genetic Operators

All operators need to be registered in a toolbox. All of these are templates. Please copy and paste them when using DEAP for the first time.

```
toolbox = base.Toolbox()
toolbox.register("expr", gp.genHalfAndHalf, pset=pset,
        min_=1, max_=2)
toolbox.register("individual", tools.initIterate,
creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)
toolbox.register("compile", gp.compile, pset=pset)
toolbox.register("evaluate", evalSymbReg, pset=pset)
toolbox.register("select", tools.selTournament,
tournsize=3)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("mutate", gp.mutUniform,
expr=toolbox.expr, pset=pset)
```

Reminder for Lexicase Selection

If you want to use lexicase selection to improve diversity, just change one line of code.

```
toolbox.register("select", tools.selAutomaticEpsilonLexicase)
```

However, keep in mind to change the fitness function to return multiple fitness values. In other words, return the error vector rather than the mean error.

```
return tuple((func(x) - x**2)**2)
```

Speedup Lexicase Selection by Numba

Numba

- Lexicase selection is hard to implement with Numpy. However, we can use Numpy to compile Python code to C code.
- You can write a Python function and simply add an annotation @njit to compile it.
- Numba needs code to only use basic Python functions. Lexicase selection in DEAP cannot be compiled by Numba.
- But, there is an out-of-the-box lexicase selection speedup by Numba in the Python package EvolutionaryForest.

```
@njit(cache=True)
def selLexicase(individuals, k):
        ...
```

Reminder for Crossover

The default GP crossover in DEAP randomly chooses the crossover point but excludes the root node (node 0).

```
list(range(1, len(ind1)))
list(range(1, len(ind2)))
```

Thus, if you really want to strictly follow the original GP, please change it to randomly select starting from 0.

```
list(range(0, len(ind1)))
list(range(0, len(ind2)))
```

Statistical Information

In DEAP, there is a built-in function to calculate simple statistical information during the evolution process.

```
stats_fit = tools.Statistics(lambda ind: ind.fitness.values)
stats_size = tools.Statistics(len)
mstats = tools.MultiStatistics(fitness=stats_fit,
size=stats_size)
mstats.register("avg", numpy.mean)
mstats.register("std", numpy.std)
mstats.register("min", numpy.min)
mstats.register("max", numpy.max)
```

Evolution Paradigm

- In most cases, eaSimple is good to use. However, if you want to use environmental selection, such as NSGA-II, you need to slightly change a few lines of code.
- Moreover, eaSimple does not keep elites. Fortunately, DEAP provides an external archive, HallOfFame, where you can add individuals from HallOfFame to the population before parent selection by changing a few lines of code.

```
population = toolbox.population(n=300)
hof = tools.HallOfFame(1)
pop, log  = algorithms.eaSimple(population=population,
        toolbox=toolbox, cxpb=0.5,
        mutpb=0.2, ngen=50, stats=mstats,
        halloffame=hof, verbose=True)
```

Ensemble Learning

- To get 20 models for ensemble learning, just change the external archive size to 20.
- After evolution, the external archive keeps the top 20 individuals, which can be used for ensemble learning.

```
hof = tools.HallOfFame(20)
```

Reminder of VarAnd/VarOr

- The default eaSimple uses the VarAnd variation scheme. It means it checks the probability of crossover first, then checks the probability of mutation. Crossover and mutation can be applied to one individual.
- John Koza's GP uses VarOr, which thus needs to change one line of code in eaSimple if you want to follow John Koza's GP.

```
offspring = varOr(offspring, toolbox, len(offspring),
cxpb, mutpb)
```

Thanks for listening!

Email: Hengzhe.zhang@ecs.vuw.ac.nz

GitHub Project: https://github.com/hengzhe-zhang/DEAP-GP-Tutorial