Jayant Parmar
Postgraduate Diploma in Artificial Intelligence
Artificial Intelligence
Indian Institute of Technology Jodhpur, Jodhpur, India
Email: g25ait1072@iitj.ac.in

Genetic Algorithm

Q1A: Roulette-Wheel Selection Modification

In the original code, tournament selection was used. As required, I made minimal and strictly necessary code updates by replacing only the selection() logic with a roulette-wheel mechanism.

Below is the modified portion of the code:

MODIFIED CODE SNIPPET — roulette_wheel_selection()

```
    def roulette_wheel_selection(self, k=None):
        population = self.population
        pop_size = len(population)
        if pop_size == 0:
            return []
        if k is None:
            k = self.population_size
```

```
    fitnesses = np.array([float(getattr(ind, 'fitness', 0.0)) for ind in population], dtype=float)

    if not np.all(np.isfinite(fitnesses)):
        fitnesses = np.nan_to_num(fitnesses, nan=0.0, posinf=0.0, neginf=0.0)

    min_f = fitnesses.min()
    if min_f <= 0:
        fitnesses = fitnesses - min_f + 1e-8

    total = fitnesses.sum()
    if total <= 0:
        probs = np.ones(pop_size) / pop_size
    else:
        probs = fitnesses / total

    indices = np.random.choice(pop_size, size=k, replace=True, p=probs)

    return [deepcopy(population[i]) for i in indices]
```

JUSTIFICATION

Roulette-wheel selection assigns proportional probabilities:

$$P(i) = f_i / \Sigma f_j$$

This ensures chromosomes with higher fitness are selected more frequently, unlike tournament selection which uses ranking. Here minimal code change was maintained — only selection logic replaced. Non-positive fitness values are shifted to positive domain without altering proportionality.

This is necessary because GA selection requires strictly positive weights for probability sampling. Numerical stability (EPS = 1e–8) avoids division-by-zero and ensures reproducibility.

Q2B: Modified Fitness Function With Parameter-Based Penalty

The original fitness function considered:

fitness = accuracy – λ * (total_parameters / scale)

The updated requirement is to:

1. Separate Conv-block parameters and FC-layer parameters.
2. Assign different weights based on computational cost.

Below is the modified portion of the code:

MODIFIED CODE SNIPPET — evaluate_fitness()

```
conv_params = 0
fc_params = 0
other_params = 0

for module in model.modules():
    if isinstance(module, (nn.Conv1d, nn.Conv2d, nn.Conv3d)):
        for p in module.parameters(recurse=False):
            conv_params += p.numel()
    elif isinstance(module, (nn.Linear,)):
        for p in module.parameters(recurse=False):
            fc_params += p.numel()
    else:
        for p in module.parameters(recurse=False):
            other_params += p.numel()

conv_M = conv_params / param_scale
fc_M = fc_params / param_scale

penalty = (w_conv * conv_M) + (w_fc * fc_M)

fitness = best_acc - (lambda_penalty * penalty)
```

JUSTIFICATION FOR WEIGHTS

Convolution layers are computationally heavier than FC layers.

Conv-layer FLOPs ≈ k*k*C_in*C_out*H*W

FC-layer FLOPs ≈ input_dim * output_dim

Hence:

- Conv blocks dominate computation → assign higher weight
- FC layers require significantly fewer operations → lower weight

Chosen weights:

    w_conv = 0.7
    w_fc = 0.3

These preserve the intent: architectures with excessive convolution parameters get proportionally penalized, while still allowing flexibility in FC layers.

Penalty Term:

    penalty = w_conv * (conv_params / scale) + w_fc * (fc_params / scale)

Final fitness:

    fitness = Accuracy – λ × penalty

This keeps accuracy primary but discourages unnecessarily large models.

LOGGING CHANGES

The revised code prints:

- conv_params
- fc_params
- penalty
- final fitness
