Team ML_ShibaInu

Group Members:

1. 2211649 - Trần Đình Đăng Khoa
2. 2252938 - Phan Chí Vỹ
3. 2252734 - Nguyễn Đức Tâm
4. 2252338 - Huỳnh Kiệt Khải
5. 2252289 - Hà Tuấn Khang

Homework 5: Genetic Algorithms

Exercise 9.1

Design a genetic algorithm to learn conjunctive classification rules for the *PlayTennis* problem (Chapter 3). Describe the bit-string encoding of hypotheses and the set of crossover operators.

1. Bit-String Encoding of Hypotheses

In the *PlayTennis* dataset, each training example has the attributes:

- Outlook ∈ {Sunny, Overcast, Rain}
- Temperature ∈ {Hot, Mild, Cool}
- Humidity ∈ {High, Normal}
- Wind ∈ {Weak, Strong}
- PlayTennis ∈ {Yes, No} (target)

A *conjunctive rule* can be represented as:

$$(\text{Outlook} = ?) \land (\text{Temperature} = ?) \land (\text{Humidity} = ?) \land (\text{Wind} = ?) \rightarrow \text{PlayTennis} \in \{\text{Yes}, \text{No}\}.$$

Bit-String Representation

We encode each attribute condition with bits indicating which value(s) are allowed:

- **Outlook** (3 possible values): `Sunny, Overcast, Rain`. We can use a 3-bit segment:

  $$[o\_Sunny, o\_Overcast, o\_Rain]$$

  A bit of 1 means "include" that attribute value, and 0 means "exclude." For example, `100` means `Outlook=Sunny`, `010` means `Outlook=Overcast`, and `110` means `(Sunny or Overcast)`.
- **Temperature** (3 values): `Hot, Mild, Cool`, a 3-bit segment.
- **Humidity** (2 values): `High, Normal`, a 2-bit segment.
- **Wind** (2 values): `Weak, Strong`, a 2-bit segment.
- **Decision Class**: Use a **1-bit** for `Yes/No` classification.

An example chromosome might look like:

$$[100, 010, 10, 10, 1]$$

This can be decoded as:

- `Outlook`: Sunny only
- `Temperature`: Mild only
- `Humidity`: High only
- `Wind`: Weak only
- `Decision`: `Yes`.

2. Crossover Operators

We adopt standard **single-point** or **two-point** crossover on the 11-bit string:

- **Single-point crossover**: choose an index k between 1 and 10, and swap the bits *after* that index between two parent chromosomes.
- **Two-point crossover**: choose two indices (k_1, k_2); swap the bit segments between k_1 and k_2.

For instance, with a single-point crossover at position 5, we swap everything after the 5th bit.

Example:

- **Parent A**: `100 010 10 10 1`
- **Parent B**: `011 100 01 00 0`
- **Offspring A**: `100 010 01 00 0`
- **Offspring B**: `011 100 10 10 1`

Exercise 9.2

Implement a simple GA for Exercise 9.1. Experiment with population size p, fraction r replaced, and mutation rate m. Show how selection, crossover, and mutation would work, and how the best solution evolves.

Implementation Outline

1. Initialize Population:

- Choose a population size p, e.g. 4.
- Generate p random 11-bit chromosomes.

2. Fitness Evaluation:

- For each chromosome, interpret it as a conjunctive rule (which attribute values are allowed).
- Compare predicted vs. actual `PlayTennis` and compute accuracy.

3. Selection Operator:

- **Tournament selection** with size 2: pick two random chromosomes, keep the one with higher fitness.
- Repeat until we select enough parents for next generation.

4. Crossover:

- For each pair of parents, pick a single crossover point in [1..10].
- Swap bits after that point to form two offspring.

5. Mutation:

- With probability m (e.g. 5%), flip each bit (0 to 1, or 1 to 0).

6. Replacement

- Keep the best individual from old population (elitism).
- Replace fraction r of the population with new offspring.

7. Termination:

- After G generations, or if the fitness stops improving for consecutive generations, stop.

Example (Two Generations)

Population Size p=4, **Mutation Rate** m=0.05, **Fraction replaced** r=0.5.

Initial Population

Chromosome	Decoded Rule	Accuracy
C_1: 100 010 10 10 1	(Outlook = Sunny) ∨ (Temp = Mild) ∨ (Humidity = High) ∨ (Wind = Weak) → Yes	0
C_2: 110 001 10 11 0	(Outlook = Sunny ∨ Overcast) ∨ (Temp = Cool) ∨ (Humidity = High) ∨ (Wind = Weak ∨ Strong) → No	0
C_3: 001 110 00 10 1	(Outlook = Rain) ∨ (Temp = Hot ∨ Mild) ∨ (Wind = Weak) → Yes	1.0
C_4: 111 010 01 11 1	(Outlook = Sunny ∨ Overcast ∨ Rain) ∨ (Temp = Mild) ∨ (Humidity = Normal) ∨ (Wind = Weak ∨ Strong) → Yes	1.0

Selection + Crossover

- Compare C_1 vs. C_3: C_3 wins (higher fitness).
- Compare C_2 vs. C_4: C_4 wins.
- Cross C_3 and C_4: single-point crossover at bit 5 yields two offspring.

Mutation

- Each bit flips with probability m=0.05, so we might see 1–2 bits changed randomly in the offspring.

Updated Population

Chromosome	Decoded Rule	Accuracy
C_3: 001 110 00 10 1	(Outlook = Rain) ∨ (Temp = Hot ∨ Mild) ∨ (Wind = Weak) → Yes	1.0
C_4: 111 010 01 11 1	(Outlook = Sunny ∨ Overcast ∨ Rain) ∨ (Temp = Mild) ∨ (Humidity = Normal) ∨ (Wind = Weak ∨ Strong) → Yes	1.0
Off1: 001 110 01 11 1	(Outlook = Rain) ∨ (Temp = Hot ∨ Mild) ∨ (Humidity = Normal) ∨ (Wind = Weak ∨ Strong) → Yes	1.0
Off2: 110 010 00 10 1	(Outlook = Sunny ∨ Overcast ∨ Rain) ∨ (Temp = Mild) ∨ (Wind = Weak) → Yes	0.66

Partial replacement can keep the best solutions. Recompute accuracy, continue to next generation. Over several generations, the **best rule emerges**.

Conclusion on GA Behavior

By tuning **population size** p, **replacement fraction** r, and **mutation rate** m, we can see:

- **Large p** → more diversity, but higher computation.
- **Higher r** → more exploration each generation, but risk losing good solutions.
- **Higher m** → more random flips, possibly helpful but can disrupt good solutions.

The best parameters typically yield a stable, high-accuracy rule that selects the right combination of attribute values for `PlayTennis`.

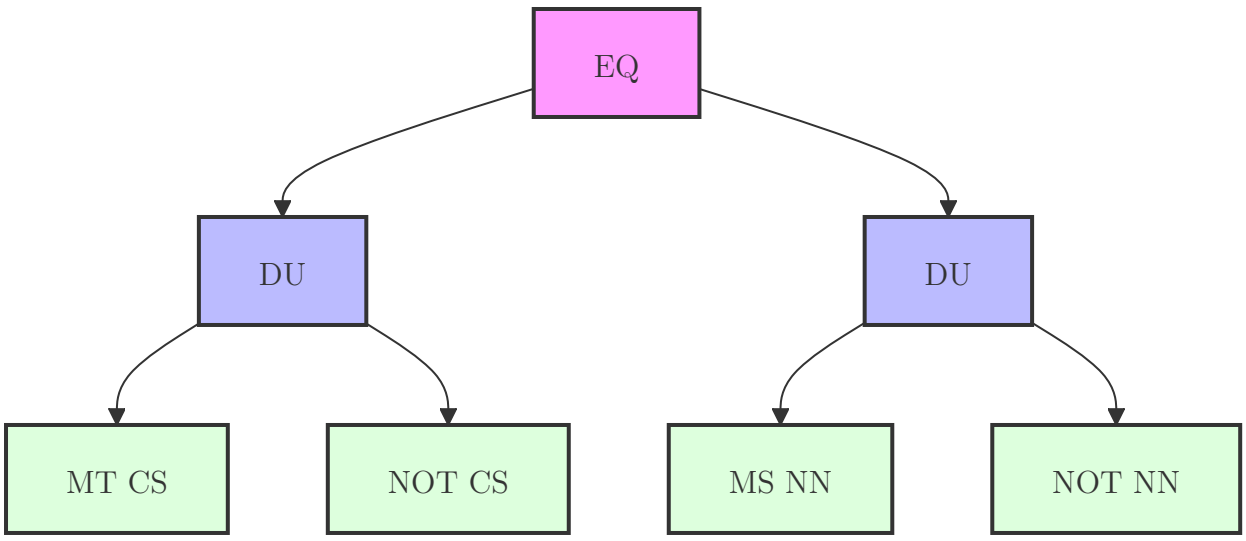
Exercise 9.3

Represent the program discovered by the GP (described in Section 9.5.2) as a tree. Illustrate the operation of the GP crossover operator by applying it using two copies of your tree as the two parents.

1. GP Program as a Tree

Koza's final discovered program for the block-stacking task (Section 9.5.2) is: `(EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))`

Here is how we can parse it into a tree structure:



Explanation of the Primitives:

- `EQ x y`: Returns T if x equals y, else F.
- `DU a b` ("Do Until"): Repeatedly execute expression a until expression b evaluates to T.
- `MT CS` (Move Top to table if `CS` is on stack): Moves the top block of the stack to the table.
- `NOT CS`: Returns T if `CS` is F, else F.
- `MS NN` (Move block `NN` to stack):
  - `NN` = Next-necessary block for building "universal."
  - If `NN` is on the table, MS moves it to the top of the stack.
- `NOT NN` checks if `NN` is F or T.

Hence the program `(EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))` effectively has:

1. A first **Do Until** loop that keeps moving the top block of the stack onto the table until the stack is empty (i.e. `NOT CS` becomes true).
2. A second **Do Until** loop that repeatedly moves the next needed block from the table to the stack until no more blocks are needed (`NOT NN`).

Finally, `EQ` wraps those two `DU` expressions, providing a valid parse.

2. Demonstrating GP Crossover Using Two Copies of the Same Tree

To illustrate crossover, we take two identical copies of this parse tree as "Parent 1" and "Parent 2." The GP crossover operator:

- Randomly selects a subtree in Parent 1,
- Randomly selects a subtree in Parent 2,
- Swaps those subtrees entirely.

Parents (Identical Trees):

Parent A: `(EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))`

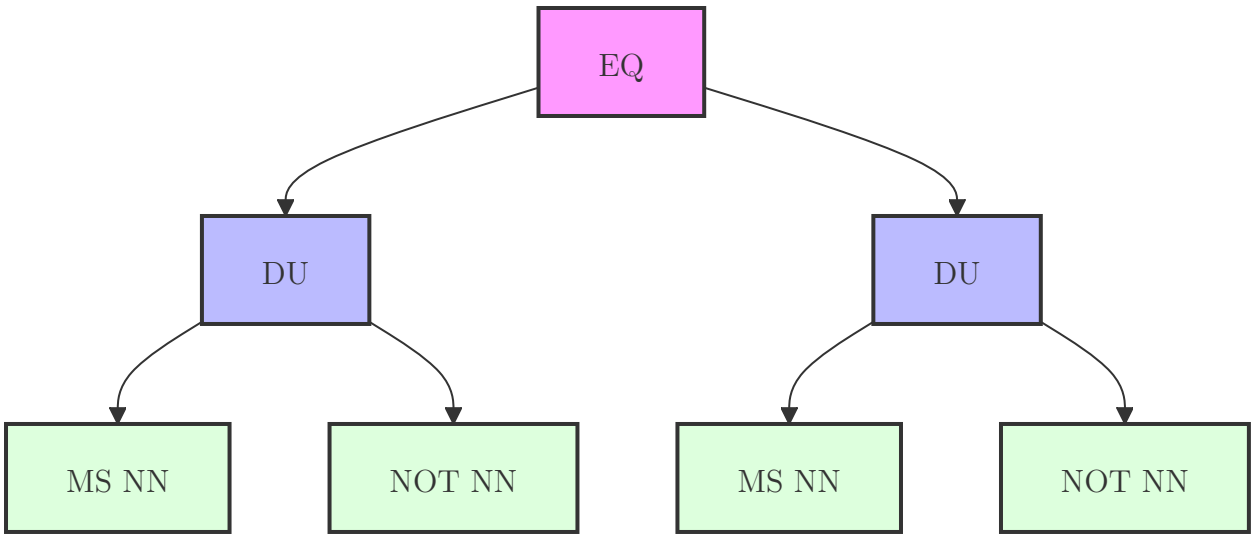
Parent B: `(EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))`

Crossover Example:

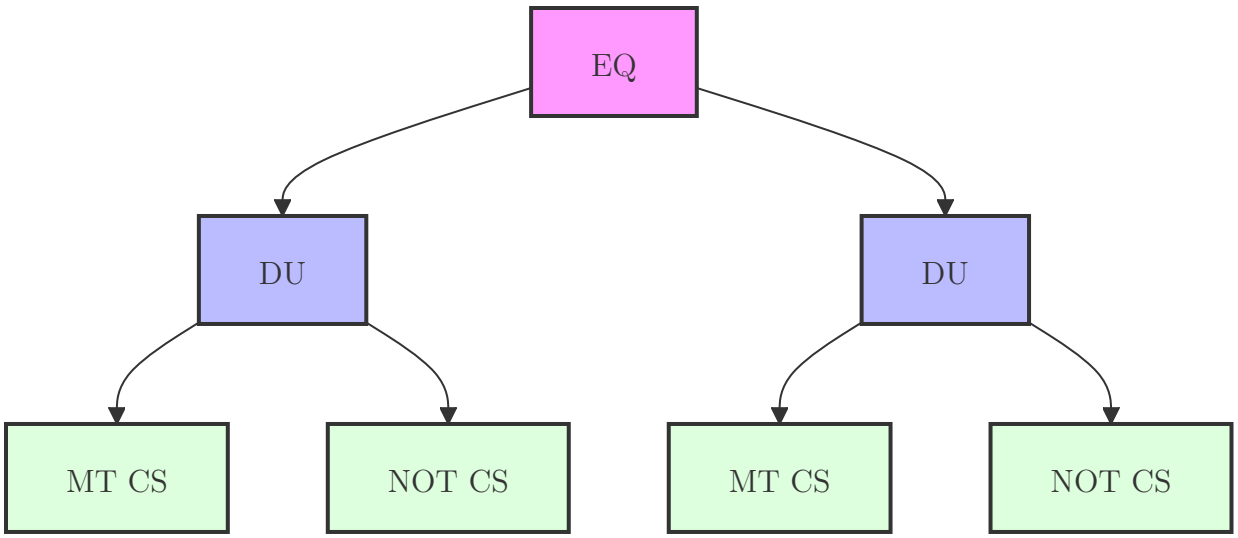
1. Randomly pick the subtree `DU (MT CS) (NOT CS)` in Parent A.
2. Randomly pick the subtree `DU (MS NN) (NOT NN)` in Parent B.
3. Swap these subtrees.

Visual Representation:

Offspring 1:



Offspring 2:



Interpretation

- **Offspring 1:** `(EQ (DU (MS NN) (NOT NN)) (DU (MS NN) (NOT NN)))` Both `DU` sub-expressions execute `MS(NN)` (move the next needed block to the stack) until `NOT NN` returns true (no more blocks needed).
- **Offspring 2:** `(EQ (DU (MT CS) (NOT CS)) (DU (MT CS) (NOT CS)))` Both `DU` sub-expressions execute `MT(CS)` (move the current top block from the stack to the table) until `NOT CS` returns true (the stack is empty).

Exercise 9.4

Consider applying genetic algorithms (GAs) to the task of finding an appropriate set of weights for an artificial neural network (in particular, a feedforward network identical to those trained by Backpropagation in Chapter 4). Consider a 3×2×1 layered, feedforward network. Describe an encoding of network weights as a bit string, and describe an appropriate set of crossover operators. Do not allow all possible crossover operations on bit strings. State one advantage and one disadvantage of using GAs in contrast to Backpropagation to train network weights.

1. The 3×2×1 Feedforward Network

We have:

- 3 **input** neurons (x_1, x_2, x_3),
- 2 **hidden** neurons (h_1, h_2),
- 1 **output** neuron (o).

Hence, there are 6 weights from input to hidden, 2 hidden biases, 2 hidden-to-output weights, and 1 output bias — totaling 11 scalar parameters.

2. Chromosome Encoding

We encode the weights *directly* as real numbers:

$$\text{Chromosome} = [\, w_{h1,1}, w_{h1,2}, w_{h1,3}, w_{h2,1}, w_{h2,2}, w_{h2,3}, b_{h1}, b_{h2}, w_{o,h1}, w_{o,h2}, b_o \,].$$

Each entry is a floating-point value (e.g. 0.23, -1.5), thus forming a vector of length 11.

Example:

$$[\, 0.45,\ -0.31,\ 1.02,\ 0.67,\ -0.25,\ 0.05,\ 0.10,\ 0.05,\ 0.90,\ 0.12,\ -0.03 \,].$$

3. Crossover Operators for Real-Valued Chromosomes

1. **Single-Point Crossover on Genes:**
   - Randomly pick an index k (1 to 10).
   - Swap the *segment* of genes from k+1 to 11 between two parent chromosomes.
   - Preserves contiguous blocks of parameters from each parent.
2. **Arithmetic Crossover**:
   - For each gene i, produce offspring genes via a weighted average:

$$w_i^{(\text{child})} = \alpha \cdot w_i^{(\text{parent1})} + (1 - \alpha) \cdot w_i^{(\text{parent2})}$$

   - $\alpha \in [0,1]$ is chosen randomly per gene or per chromosome.

4. Mutation for Real-Coded GA

Since weights are real numbers, we use:

- **Gaussian Mutation**: For each gene i, with probability m, do w_i ← w_i + Δ, Δ ~ N(0,σ^2).
- **Uniform Mutation**: For each gene i, with probability m, re-sample w_i from some allowed range, e.g. [-1,1].

5: Advantage & Disadvantage of Using GA vs. Backpropagation

Advantage (GA)

- **Global Search:** Genetic Algorithms explore a broader parameter space, making them *less likely to get trapped* in local minima. They can also work with non-differentiable activation or loss functions.

Disadvantage (GA)

- **Slower Convergence:** GAs require evaluating a *population* of solutions each generation, which is often *computationally more expensive* than the direct gradient-based approach of Backpropagation. This can be significant especially for large networks.