CSG2341 Intelligent Systems

Module 11: Hybrid Intelligent Systems

The Intelligent Systems library can be used to build some kinds of hybrid intelligent systems. This document explains how to do it.

Evolving Neural Networks

In Negnevitsky Section 8.5, the author describes how evolutionary algorithms can be used to evolve neural networks to solve problems, as an alternative to training schemes like back-propagation. This technique can be used to evolve network weights, or the network structure, or both.

The Intelligent Systems library can be used to evolve a set of weights for a `Perceptron` or a `MultilayerPerceptron`. This is done using an abstract class in the `evolution` package, `GeneArrayEvolvable`, which is a subclass of the `Evolvable` class.

A `GeneArrayEvolvable` object can be thought of as an array of genes, each of which can be copied or mutated. Mutation of the array then works by randomly selecting a gene to mutate, and calling its `mutate()` method. Crossover simply copies genes from one array to the other, using one- or two-point crossover, or uniform crossover.

The IS library provides a concrete subclass of `GeneArrayEvolvable` in the `neural` package, `NeuronVectorGenome`. It has two constructors

```
public NeuronVectorGenome
      (
            MultiLayerPerceptron mlp,
            Random random,
            Double mutationRate,
            Integer crossoverType
      ) throws Exception;

public NeuronVectorGenome
      (
            Perceptron perceptron,
            Random random,
            Double mutationRate,
            Integer crossoverType
      ) throws Exception;
```

The constructors both work by using a neural network as a “template”, and constructing a genome that has one gene for each neuron in the network.

Each gene is an array of the weights on the connections coming into that neuron. This is the representation that is described in Section 8.5 of your text book.

The class also has a method:

```
    public void copyToNeurons();
```

which copies the weights from the genome into the neural network. The network can then be tested on the problem to be solved.

Here is a code segment that shows how to create a genetic algorithm for evolving a neural network:

```
        Random random = new Random();
        MultiLayerPerceptron mlp = new MultiLayerPerceptron
                  (
                        new int[]{2, 5, 1},
                        new Random(), 0.1, 0.6, false
                  );
        Evaluator evaluator = new SomeEvaluator();

        // Parameters for the constructor for the Evolvable object
        //
        // The Evolvable is a set of weights for an MLP
        //
        Object[] constructorParams =
            {
                mlp,
                random,
                new Double(0.01), // mutation rate
                new Integer(EvolutionaryAlgorithm.ONEPOINT)
            };

        GeneticAlgorithm ga = new GeneticAlgorithm
            (
                NeuronVectorGenome.class,
                constructorParams,
                100,  // population size
                random,
                0.05, // mutation probability
                0.7,  // crossover probability
                evaluator,
                new GenerationsTerminator(1000),
                new StochasticUniformSelector(random),
                true
            );
```

Once the GA has been run, the evolved network can be extracted as follows:

```
      NeuronVectorGenome genome = (NeuronVectorGenome)ga.getBestEver();
      genome.copyToNeurons();    // copies genome data to mlp
```

Evolving Fuzzy Rule Sets

In Negnevitsky Section 8.6, the author describes one way to evolve a fuzzy rule set. We do it differently, again using the `GeneArrayEvolvable` class, and using an existing fuzzy rule set as a “template”. The IS library provides another concrete subclass of `GeneArrayEvolvable` in the `Fuzzy` package, `FuzzyRuleSetGenome`. It has two constructors:

```
public FuzzyRuleSetGenome
      (
            SugenoRuleSet ruleset,
            Random random,
            Double mutationRate,
            Integer crossoverType
      ) throws Exception;

public FuzzyRuleSetGenome
      (
            MamdaniRuleSet ruleset,
            Random random,
            Double mutationRate,
            Integer crossoverType
      ) throws Exception;
```

and a method

```
public void copyToRuleSet();
```

The constructors use a rule set as a template, and create a genome that is an array with two types of genes. There is one gene for each `FuzzyVariable` in the rule set, which represents the shapes of the `FuzzySets` for that variable. And there is one gene for each `FuzzySpike` that is the right-hand side of a rule in the rule set, which determines the crisp value for the spike.

The method `copyToRuleSet()` copies the shapes and values in the genome onto the `FuzzySets` and `FuzzySpikes` in the rule set. The rule set can then be tested out.

Here is a code segment showing how to create a GA to evolve a fuzzy rule set:

```
Random random = new Random();
MamdaniRuleSet ruleset = new MamdaniRuleSet();
//
// add code to define fuzzy variables, fuzzy sets, fuzzy rules
//
Evaluator evaluator = new SomeEvaluator();

// Parameters for the constructor for the Evolvable object
//
// The Evolvable is a FuzzyRuleSetGenome, which can be used
// to evolve parameters for a FuzzyRuleSet
//
Object[] constructorParams =
    {
        ruleset,
        random,
        new Double(0.01), // mutation rate
        new Integer(EvolutionaryAlgorithm.ONEPOINT)
    };

GeneticAlgorithm ga = new GeneticAlgorithm
    (
        FuzzyRuleSetGenome.class,
        constructorParams,
        100,  // population size
        random,
        0.05, // mutation probability
        0.7,  // crossover probability
        evaluator,
        new GenerationsTerminator(1000),
        new StochasticUniformSelector(random),
        true
    );
```

Once the GA has been run, the evolved rule set can be extracted as follows:

```
FuzzyRuleSetGenome genome = (FuzzyRuleSetGenome)ga.getBestEver();
genome.copyToRuleSet();    // copies genome data to ruleset
```