Evolutionary Learning, Part 2

Genetic algorithm design issues

- Representation of candidate solutions
- Fitness function
- Selection method
- Genetic operators (crossover, mutation, etc.)
- Parameters
  - Population size
  - Crossover probability
  - Mutation probability
  - Number of generations

Robby the Robot: Parameters

```
NUM_GENERATIONS        500

# Options for selection_method are "fitness proportionate", "pure
   rank", and "elite"
SELECTION_METHOD       "pure rank"

POP_SIZE               200

CHROM_LENGTH           243

MUTATION_RATE          0.005 # probability of mutation at each locus in
   a chromosome

CROSSOVER_SITES        1

CROSSOVER_RATE         1.0 # probability of two parents crossing over

NUM_ELITE              50   # fill in if selection method is "elite"
```

```
FITNESS_FUNCTION_NAME   "RobbyTheRobot"

WALL_PENALTY -5   # Points lost for crashing into a wall

CAN_REWARD 10     # Points gained for picking up a can

CAN_PENALTY -1    # Points lost for trying to pick up a can in an
   empty cell

NUM_MOVES 200     # Number of moves a robot makes per session

NUM_ENVIRONMENTS_FOR_FITNESS 200      # Number of environments each
                                      # individual is tested on for
                                      # calculating fitness
```

```
generate_population();  /* Generate random population */

for (current_generation=1; current_generation <= NUM_GENERATIONS;
   current_generation++) {

   for (current_chrom = 0; current_chrom < POP_SIZE; current_chrom++)  {
        /* Calculate fitness of next individual */
        current_pop[current_chrom].fitness =
            calc_fitness(current_pop[current_chrom].chrom, current_chrom);

   }
   assign_weights_to_population();
   weight_sum = 0.0;
   for (i=0; i < POP_SIZE; i++)
      weight_sum += current_pop[i].weight;

   /* Create next population */
   for (i=0; i < POP_SIZE/2; i++) {
        parent1 = roulette_wheel(weight_sum);
        parent2 = roulette_wheel(weight_sum);

        /* Perform crossover on parents, mutate children, and add
        children to new population */
        perform_crossover_and_mutation(parent1,parent2)
   }
   /* Current population ←——— New population */
   update_population();
}
```

Main loop of GA

Assigning weights

```
assign_weights_to_population()
{
   if (strcmp (SELECTION_METHOD, "elite") == 0)
        assign_elite_weights();
   else if (strcmp (SELECTION_METHOD, "pure rank") == 0)
        assign_pure_rank_weights();
   else if (strcmp (SELECTION_METHOD, "fitness proportionate")
   == 0)
        assign_fitness_proportionate_weights();
}
```

Elite weights

```
assign_elite_weights()
{
  int i;
  /* We assume population has been sorted in order of
   decreasing fitness */
  for (i=0; i < NUM_ELITE; i++)
    current_pop[i].weight = 1.0; /* Equal weight to each elite
                                    string */

  for (i=NUM_ELITE; i < POP_SIZE; i++)
    current_pop[i].weight = 0.0;
}
```

Pure-rank weights

```
assign_pure_rank_weights()
{
  int i;

  for (i=0; i < POP_SIZE; i++) {
    current_pop[i].weight = (POP_SIZE - i);
  }
}
```

Fitness-proportionate weights

```
assign_fitness_proportionate_weights()
{
   int i;

   for (i=0; i < POP_SIZE; i++) {
        /* Subtract min_pop_fitness to avoid negative weights,
           and add MIN_INCREMENT to avoid zero weights */
        if (min_pop_fitness < 0)
                current_pop[i].weight
                   = current_pop[i].fitness - min_pop_fitness  +
                                MIN_INCREMENT
        else
                current_pop[i].weight = current_pop[i].fitness
   }
}
```

Roulette wheel Selection

```
int roulette_wheel(double weight_sum) {
/* Returns index of chosen parent. */

  int index=-1;
  double r, sum=0;

  r = random() * weight_sum;

  while (sum < r) {
    index++;
    sum += current_pop[index].weight;
  }

  return(index);

}
```

Selection methods

- Fitness proportionate selection
- Rank selection
- Elite selection
- **Tournament selection**
  - Select a *tournament* of *t* random individuals. Do fitness-proportionate selection within that tournament to select a parent.

GA Demo

http://www.cs.pdx.edu/~mm/RobbyTheRobot

Case Study 1
Evolving image-processing algorithms

- GENIE project (GENetic Image Exploitation), Los Alamos National Laboratory

Brumby, Harvey, Perkins, Porter, Bloch, Szymanski, et al.

Consider a large library of multi-spectral photographic images from satellites and high-flying airplanes.

- How to automatically find images containing desired features?

  E.g., water, clouds, craters on Mars, forest fire damage, vegetation, urban areas, roads, airports, bridges, beaches,...

Special-purpose algorithms can (possibly) be developed for any particular feature, but goal of GENIE project is to have a general-purpose system:

A user can specify *any* feature (via training examples), and system will quickly find images containing that feature.

Some possible applications:

Remote sensing

Ecological and environmental modeling
planetary image analysis

Medical imaging

Image database search

Training Examples for GENIE

- Input images consist of ~10-100 spectral planes.
- User brings up training image(s) in GUI
- User paints examples of “true” and “false” pixels.

From Brumby et al., 2000

Figure 3. Training scene is taken from an AVIRIS overflight of NASA Moffet Field Air Station.

From Brumby et al., 2000

Human created training images

Green = “feature”

Red = “non-feature”

Black = “not classified”

Roads

Vegetation

Water

Data and Feature Planes

Data planes
(spectral channels)

Truth plane

"Feature" planes

What GENIE Evolves

- GENIE evolves an image processing algorithm that operates on data planes to produce a new set of *feature planse.*

- Feature planes (along with data planes) are used to train a conventional classifier (e.g., perceptron, SVM, etc.)

GENIE's Gene's

- *Genes* are elementary image-processing operations that read from data planes (spectral channels) or feature plane and write to feature planes

  e.g.,

  addp(d1, d2, f1, f2)

  linear_comb(d2, d1, f1, 0.9)

  erode(d1, f2, 3, 1)

- Elementary operators include spectral, morphological, logical, and thresholding operators

GENIE's Chromosomes

- *Chromosomes* are algorithms made up of genes

```
lawd (d3, f2)                    ; Law's textural operator
variance (f2, f2, 3, 1)          ; local variance in circular neighb.
min(d1, d2, f1)                  ; min at each pixel location
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)             ; textural operator r5r5
open (f2,f2, 3,1)                ; erode, then dilate
subp (d1, f1, f1)                ; subtract planes
min (d0, f2, f2)
adds (d1, f0, 0.25)              ; add scalar
opcl (f1,f1, 1, 1)               ; erode, dilate, dilate, erode
h_basin (f0, f0, 18)             ; regional minima
linear_comb (f0, d9, f0, 0.2)
linear_comb (d4, f2, f2, 0.9)
addp (d6, f0, f0)                ; add planes
```

GENIE's Population

- *Initial population* is a set of 20-50 randomly generated chromosomes:

```
awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
subp (d1, f1, f1)
min (d0, f2, f2)
adds (d1, s0, 0.25)
opcl (f1,f1, 1, 1)
```

```
erode(d3, f2,3,1)
linear_comb (d2, d1, f2, 0.5)
min (f1, f2, f2)
open (f2,f2, 3,1)
```

```
open (f1,f2, 1,1)
variance (f2, f2, 3, 1)
addp (f1, s0, f2)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)
```

...

GENIE's Fitness

- *Fitness:*
  - Run chromosome on image data to produce feature planes.
  - Find linear combination of planes that maximizes spectral separation between labeled "true" and "false" pixels.
  - Threshold resulting plane to get binary image (each pixel is 1 or 0):
    - 1 = feature, 0= non-feature
    - Use threshold that will maximize fitness

GENIE's Fitness

– Fitness:

$$F = 500(R_d + (1 - R_f))$$

where

R_d is **true positive rate**: fraction of "feature" pixels (green) classified correctly

R_f is **false positive rate**: fraction of non-feature pixels (red) classified incorrectly

F=1000 is perfect fitness.

The GA

- *Selection*: Elite selection
- *Crossover*: Recombine different genes from two different parents
- *Mutation*: Randomly change a chosen gene

Parent 1

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
subp (d1, f1, f1)
min (d0, f2, f2)
adds (d1, s0, 0.25)
opcl (f1,f1, 1, 1)

Parent 2

open (f1,f2, 1,1)
variance (f2, f2, 3, 1)
addp (f1, s0, f2)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Parent 1

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
subp (d1, f1, f1)
min (d0, f2, f2)
adds (d1, s0, 0.25)
opcl (f1,f1, 1, 1)

Parent 2

open (f1,f2, 1,1)
variance (f2, f2, 3, 1)
addp (f1, s0, f2)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Child

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Parent 1

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
subp (d1, f1, f1)
min (d0, f2, f2)
adds (d1, s0, 0.25)
opcl (f1,f1, 1, 1)

Parent 2

open (f1,f2, 1,1)
variance (f2, f2, 3, 1)
addp (f1, s0, f2)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Child

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)

smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Parent 1

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
open (f2,f2, 3,1)
subp (d1, f1, f1)
min (d0, f2, f2)
adds (d1, s0, 0.25)
opcl (f1,f1, 1, 1)

Parent 2

open (f1,f2, 1,1)
variance (f2, f2, 3, 1)
addp (f1, s0, f2)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

Child

awd (d3, f2)
variance (f2, f2, 3, 1)
min(d1, d2, f1)
linear_comb (f1, d0, f1, 0.9)
smooth_r5r5 (f2, f2)
lawd(f1,f3)
smooth_r5r5 (d1, f3)
opcl (f1,f1, 1, 3)
lawd(d2, s0)
subp (f1, f1, f1)

- The GA is run until acceptable performance on the training examples is found.

- Generalization performance is evaluated, and user then creates new training example, based on generalization performance.

Some Results

From Brumby et al., 2000

Figure 3. Training scene is taken from an AVIRIS overflight of NASA Moffet Field Air Station.

From Brumby et al., 2000

Figure 4. Analyst supplied training data (on left) and evolved tool output (on right) for, from top: roads/buildings, vegetation, water. "True" pixels appear light grey, "false" pixels appear as medium grey. Unclassified pixels appear black.

From Brumby et al., 2000

Figure 6. Result of application of the evolved tools to a new scene from the same AVIRIS campaign.

Recent work by Ghosh & Mitchell

- Combine GENIE's texture-learning ability with genetic shape-learning algorithm.
- Application: Find prostate in CAT scans of pelvic region

From Ghosh & Mitchell, 2006

Training image, contour drawn by radiologist

(Prostate pixels painted green for GENIE input; samples of non-prostate pixels painted red)

Test image, with hand-segmentation by a radiologist

Results from GENIE alone

Results from our hybrid algorithm

Case Study 2
Genetic Art and Computer Graphics
(Karl Sims, 1993)

- See http://www.hedweb.com/biomorph/biomorph.htm
- GA individuals: equations that generate a color for each pixel coordinate
- Function set:

  +, -, /, mod, round, min, max, abs, expt, log, and, or, xor, sin, cos, atan, if, dissolve, hsv-to-rgb, vector, transform-vector, bw-noise, color-noise, warped-bw-noise, warped-color-noise, blur, band-pass, grade-mag, grad-dir, bump, ifs, warped-ifs, warp-abs, warp-rel, warp-by-grad

- Terminal set:

  X, Y, rand_scalar(), rand_vector(n)

- Each function returns an image (an array of pixel colors)
- Could add a third dimension Z to allow volume, and a fourth dimension T to allow time (animations)

Left to right, top to bottom:

a. *Y*
b. *X*
c. *(abs X)*
d. *(mod X (abs Y))*
e. *(and X Y)*
f. *(bw-noise .2 2)*
g. *(color-noise .1 2)*
h. *(grad-direction (bw-noise .15 2) 0 0)*
i. *(warped-color-noise (* X .2) Y .1 2)*

Some Results

```
(round (log (+ y
(color-grad (round
(+ (abs (round (log
(+ y (color-grad
(round (+ y (log
(invert y) 15.5)) x)
3.1 1.86 #(0.95 0.7
0.59) 1.35)) 0.19)
x)) (log (invert y)
15.5)) x) 3.1 1.9
#(0.95 0.7 0.35)
1.35)) 0.19) x)
```

http://mzlabs.com/MZLabsJM/page4/page22/page22.html

Evolving Virtual Creatures
Sims, 1994

- Goal is to evolve life-like simulated “creatures”.
- Simultaneously evolve morphology and control systems of simulated creatures that can “walk”, swim, jump, follow light.
- Related to Dawkins’ Blind Watchmaker program, which evolved morphologies.
- Environment is detailed simulation of physics.

Creature morphology

- Morphology developed from directed graph of nodes and links.
- Each node of the graph: body part and neural network controlling that body part

Genotype graphs and corresponding morphologies

Creature control

- Neural network controlling each body part:
  - Function that inputs sensor values and outputs effector values (forces or torques at joints)

- Sensors:
  - Joint-angle sensors: returns value of angle
  - Contact sensors: 1.0 if contact is made (with anything), -1.0 if no contact
  - Photosensors: return coordinates of light source direction relative to orientation of part

Creature control

- Neurons: can have different activation functions (function of inputs):
  - sum, product, divide, sum-threshold, greater-than, sign-of, min, max, abs, if, interpolate, sin, cos, atan, log, expt, sigmoid, integrate, differentiate, smooth, memory, oscillate-wave, oscillate-saw.

Body phenotype

Genotype

Neural network phenotype

Creature control

- At each simulated time step, every neuron computes its output value from its inputs.
  - Two brain time steps performed for each dynamic simulation time step.

- Effectors: get input from a single neuron or sensor, output joint force (like a muscle force).
  - Each effector controls a degree of freedom of a joint.

Physical Simulation

- Complicated physical simulation is used:
  - articulated body dynamics, numerical integration, collision detection, collision response, friction, viscous fluid effect.
- Parallel implementation on a Connection Machine

Evolution of Creatures

- Creature grown from its genetic description.
- Run in simulation for some period of time
  - Sensors provide data about world and creature's body to brain
  - brain produces effector forces which move parts of creature
  - Fitness: how successful was desired behavior by end of allotted time

Video:
http://www.youtube.com/watch?v=JBgG_VSP7f8

Fitness

- Swimming:
  - Environment: Turn off gravity, turn on viscous water effect.
  - Fitness: Swimming speed (distance of center of mass/time from initial center of mass position)
    - Special tweaks to prevent circling, initial push versus constant motion
    - Notice that tweaks on all of these make clear the trial and error aspect of this project
- Walking:
  - Environment: Turn on gravity, turn off viscous water effect.
  - Fitness: Horizontal speed.
    - Tweak: To prevent tall creatures that simply fall over, run simulation with no friction and no effector forces until height of center of mass reaches a stable minimum.

Fitness

- Jumping:
  - Environment: Turn on gravity, turn off viscous effect.
  - Fitness: Maximum height above the ground of lowest part of creature.
-
- Following a light source:
  - Environment: Enable photosensors.
  - Fitness: Average speed following different light sources over several trials.

Evolution of Creatures

- Create initial population of genotypes
- Best 20% survives. Rest are replaced by offspring of best 20%.
- Mutation:

  Many different types, depending on object, e.g.:

  - Numerical parameters changed by small random amounts
  - New random nodes are added
  - Connections randomly moved
  - New random connections generated