Neural and Evolutionary Computing.

Lab 6: Evolution strategies. Genetic programming. Evolutionary training of neural networks.

---

1. Implementation of Evolution Strategies

In the case of evolution strategies the elements of the population are real vectors and the main components are:

- *Selection:* it is used only to select the survivors (all elements can be parents) and it is usually a deterministic selection based on taking the best M offspring from the set of L offspring (in the case of (mu,lambda) strategies) or the best M elements from the joined population of parents and offspring (in the case of (M+L) variants). M denotes the number of elements in the current population and L denotes the number of elements generated using recombination and mutation.
- *Recombination:* from rho parents is constructed one offspring by linear (convex) combination.
- *Mutation:* it is applied to all elements in the population and consists of adding a random value (generated according to a given distribution).

Application 1. Function minimization. Test functions: sphere and Griewank.
See for instance:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm

Hint: an example is implemented in SE.sci

Exercises:

1. Test SE.sci for Ackley, Rastrigin and Rosenbrock functions described in the web page http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm

2. Genetic programming.

Genetic programming aim is to design in an evolutionary manner computational structures (arithmetical/logical expressions, classification/decision rules or programs). In traditional Genetic Programming applications (as symbolic regression) the population elements are hierarchical structures (e.g. syntactic trees). The genetic operators are adjusted to work with such structures. One of the main difficulties in GP is to avoid the proliferation of large structures (the so called bloat problem). A possible solution to this problem is to limit the depth of the trees generated during the evolutionary process.

The most popular application of GP is symbolic regression aiming to evolve an expression which fits well to some data (unlike the numerical regression which aims to estimate the coefficients of a given model, symbolic regression estimates the model itself).

Application 2a. Evolve expression which fit some datesets using the Java applet available at http://www.geneticprogramming.org/symbolic/main.htm

Main steps:

- *Choice of the test function* (Function Settings): specify the definition domain (Min X, Max X), the function to be approximated (Enter Function) and the number of elements in the dataset to be used to evaluate the quality of an evolved expression (Number of points)
- *Setting the parameters of the algorithm*: number of generations, population size, maximal depth of the trees in the initial population, croassover fraction, mutation fraction, maximal depth of the trees constructed through crossover, maximal depth of the subtrees used in mutation
- *Choosing the population initialization style:*
  - Full: all branches in the tree have the maximal depth
  - Grow: the extension of the branches is stopped (based on a random event) before reaching the maximal depth
  - Ramped half-half: half of the population elements are generated using the Full style,while the other half is generated using the Grow style.
- *Choosing the selection method*:
  - Proportional
  - Tournament
- *Choosing the set of nonterminals* (by marking the operators/functions available in the list: +,-,*,/, sin,cos,exp, abs, max, min, log)
- *Choosing the set of terminals* (variable x and rand - to generate random values)

Application 2b. Use the "rgp" R package to find an expression which fits a dataset.

Main steps:

- Launch R
- Load package "rgp": Packages ->Load package … or library("rgp") (if the package is not installed then it should be installed by Packages-> Install package(s)…
- Define the set of nonterminals (operators and functions) using functionSet. Example: setNonterminals <- functionSet("+", "*", "-","/")
- Define the set of variables using inputVariableSet. Example: setVariables <- inputVariableSet("x")
- Define the set of constants constantFactorySet. Example: setConstants <- constantFactorySet(function() rnorm(1)) (random values generated using the standard normal distribution)
- Define the test data: values which will be used to evaluate the approximation accuracy. Examplu: dateX <- seq(from = -pi, to = pi, by = 0.1)
- Define the fitness function: mean square error (measure of the difference between the values of the test function and the values corresponding to the evolved expressions). Example: fitness <- function(f) rmse(f(dateX), sin(dateX))
- Call the function corresponding to the evolutionary process (geneticProgramming). Example: geneticProgramming(functionSet = setNonterminals,inputVariables = setVariables,constantSet = setConstants, fitnessFunction = fitness,stopCondition = makeStepsStopCondition(10000))

Particularities of the genetic programming implemented in "rgp":

- The population elements are R expressions (implemented as tree-like structures)
- The population initialization is based on several construction strategies:

- "grow" (each branch in the tree will be extended until it reaches the maximal length or until a random event occurs)
- „full" (all branches in the tree have the maximal length)
- Combined variant (some elements are generated using the "grow" strategy, others are constructed using the „full" strategy)

- The package implements the traditional crossover and mutation strategies adapted for trees (see slides of lecture 9)
- There are implemented various selection variants using one or several criteria (as in multiobjective optimization). In the multicriterial variant the aim is to optimize the quality of the result, the simplicity of the elements and the population diversity.

Exercise: Follow the above steps and test the influence of nonterminals on the quality of the results. Hint: gp1.r

3. Evolutionary training of neural networks

EAs can be used for neural networks design at least for two problems:

- Estimation of the synaptic weights in the case of networks with non-differentiable activation/transfer functions or in the case of recurrent networks (when traditional algorithms like Backpropagation cannot be applied)
- Establishing the architecture of the network.

Application 2. Let us consider the problem of training a neural network in order to represent the XOR function.
We can use the following architecture:

- 2 input units + 1 dummy unit (used to model the biases for the hidden units)
- K hidden units (K is an input parameter) with a Heaviside activation function
- An output unit with a Heaviside activation function

Each element of the population will have 4*K+1 components corresponding to all weights and biases. The function to be minimized is the MSE (Mean Squared Error) computed for the training set containing all four examples: ((0,0),0), ((0,1),1),((1,0),1), ((1,1),0). Analyze the influence of the population sizes, mutation parameter, number of parents used for recombination, selection type (truncation or tournament) and value of K.

Hint. Use SE_nn.m. The function used by the evolution strategy to evaluate a network is SE_XOR.

Homework

1. Change the evolution strategy implemented in SE.sci by including self-adaptation (see lecture 8)
2. Extend the evolutionary training of a neural network (for XOR representation) for the case when the number of hidden units is variable (the population elements will contain a component corresponding to the number of hidden units before the components used for storing the weights. The length of each component will be 4Kmax+1 (Kmax is the maximal number of hidden units) and for a given value K<Kmax only the first 4K+1 weights will be used.