Organization & Search Algorithms

Machine Learning B
708.062 08W 1sst KU

Dr. Stefan Häusler
Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz

Course WS 2010/11
Course information

Please register for 708.061 (lecture) and 708.062 (exercises) via TUGOnline.

Further informationen:

- Web page: http://www.igi.tugraz.at/lehre/MLB
- Newsgroup: tu-graz.lv.maschinelles-lernen
Assignments and Grading

A part of the assignments will be **programming tasks** (MATLAB), and the rest will be **theoretical problems**.

Your grade for the exercises is determined by two components:

1. Points for the assignments (100%).
2. Additional bonus points (optional) on the assignments (approx. 20%)
Course schedule

Assignments will be handed out every three weeks followed by a lecture (explanation hour) after one week. Solutions have to be handed in two weeks afterwards at the beginning of presentation hours.

- In presentation hours for every exercise one student is selected to present his/her solution on the board.

- Every student has to present at least one solution during the semester.

Hand out assign.  Hand in assign.
Teams

You may form teams of two students to discuss and work together on the problems. Four rules have to be obeyed:

1. You must **name your team members** on the first page of your submitted solution.

2. Everybody must submit his/her own solution and write down everything in **his/her own words**.

3. For programming examples a team may use the **same MATLAB code**, but nevertheless everybody must write his/her own report.

4. Each member of the team must **be able to present** the solution on the blackboard.
Copying assignments

Copying of solutions or parts of solutions of other teams is not allowed!

Code sharing between teams is not allowed!
Assignment cover

Each student should use for each assignment (number) a separate cover.

Assignment __

Machine Learning B
708.062 10W KU

<table>
<thead>
<tr>
<th>Last name</th>
<th>First name</th>
<th>Matriculation number</th>
<th>Team members</th>
</tr>
</thead>
</table>

Available at: http://www.igi.tugraz.at/lehre/MLB/WS10/benotung/cover.pdf
First homework


Comparison of different (heuristic) search methods:

- Genetic Algorithms
- Particle Swarm Optimization
- Simulated Annealing
Assignment 1

1 Comparison of Optimization Algorithms [5 P]

Apply the four optimization algorithms gradient descent, genetic algorithms, simulated annealing, and particle swarm optimization to minimize each of the following four 2-dimensional functions:

Rastrigin

Rosenbrock

Ackley

Chasm
Genetic Algorithm Toolbox GAOT

Old but simple genetic algorithm toolbox in Matlab
(another option is ga.m)

See gaotv5.pdf for help.

See GenAlgEx.m for an example.
GAOT Components

- Evaluation function
- Selection function
- Reproduction functions: Mutation and crossover
- Definition of the initial population and the stopping criterion
Pseudo-code

- $i = 1$
- repeat
  - $P_i' = \text{select}(P_{i-1})$
  - $P_i = \text{reproduce}(P_i')$
  - evaluate($P_i$)
  - $i = i+1$
- until stopping criterion applies

Solution: individual with maximal fitness in any population $P_i$
ga.m

[x, endPop, bPop, traceInfo] =
    ga(bounds, evalFN, evalParams, params, startPop, termFN,
       termParams, selectFN, selectParams, xOverFNs,
       xOverParams, mutFNs, mutParams);

x                          ... best solution
endPop                     ... final population
bounds                     ... upper and lower bounds of genes, e.g. [l1 u1; l2 u2 ...]
evalFN                     ... Name of evaluation Function
startPop                   ... initial population, see initializega.m
termFN                     ... Stopping criterion
selectFN                   ... Name of selection function
xOverFNs                   ... String of names of XOver operators, e.g. 'name1 name2 ...'
mutFNs                     ... String of names of mutation operators, e.g. 'name1 name2 ...'
<...>Params                ... Parameters for the called function, [ #operators x #parameters]
Evaluation Function

Option 1: MATLAB function call

```matlab
function [val] = cost_function(x)
evalFN = 'cost_function(x)';
```

Option 2: MATLAB command

```matlab
evalFN = '-(x'*x);
```

Option 3: MATLAB function call (see GenAlgEx.m)

```matlab
evalFN = 'cost_function';
[x,val] = Gaeval(x,currentgeneration, ...parameters... )
```

NOTE: First two options have x as input argument.
The value of evalFN will be maximized!!
Selection Functions I

Roulette Wheel: ‘roulette’

\[ P[ \text{Individual } i \text{ is chosen } ] = \frac{F_i}{\sum_{j=1}^{Pop\ Size} F_j}, \]

\[ F_i \text{ ... Fitness of individual } i \]

- No negative fitness values allowed (only maximization)
Selection Functions II

Normalized Geometric Selection: 'normGeomSelect'

- Rank individuals according to fitness
- Negative fitness is allowed

\[
P[\text{selecting the } i^{th} \text{ individual}] = \frac{\frac{q}{1- (1- q)^P}}{(1- q)^{r-1}}
\]

- \( q \) ... Prob. of selecting the best individual
- \( r \) ... Rank of the \( i^{th} \) individual
- \( P \) ... Population size
Selection Functions III

Tournament Selection: 'tournSelect'

- Select randomly (with replacement) \(j\) individuals from the population
- Insert best of the \(j\) individuals into the new population
- Repeat until \(N\) individuals are selected
Mutation & Crossover

\[ P_1 \begin{array}{l}
111010 \\
101101 \\
101101 \\
\vdots
\end{array} \quad \rightarrow \quad \begin{array}{l}
101101 \\
\downarrow \\
001111
\end{array} \]

\[ 2 \text{ bit flips} \]

\[ P_1 \begin{array}{l}
111010 \\
101101 \\
101101 \\
001111 \\
\vdots
\end{array} \]

\[ P'_1 \begin{array}{l}
111010 \\
101101 \\
101101 \\
001111 \\
\vdots
\end{array} \]

\[ \begin{array}{l}
111010 \\
101101 \\
101101 \\
001111
\end{array} \quad \rightarrow \quad \begin{array}{l}
111010 \\
101101 \\
111101 \\
101010
\end{array} \]

\[ \begin{array}{l}
111010 \\
101101 \\
101101 \\
001111 \\
111101 \\
101010
\end{array} \]
Mutation Operators I

**Binary Mutation:** Random bit-flips

- 'binaryMutation'
- $100101 \rightarrow 101101$

**Uniform Mutation:**

- Randomly select one variable $j$ of the string and set it to an uniform random number in the interval $[a_i, b_i]$
- 'unif Mutation'

$$x'_i = \begin{cases} 
U(a_i, b_i), & \text{if } i = j \\
x_i, & \text{otherwise}
\end{cases}$$
Mutation Operators II

Boundary Mutation:

- 'boundaryMutation'

\[
x'_i = \begin{cases} 
  a_i, & \text{if } i = j, r < 0.5 \\
  b_i, & \text{if } i = j, r \geq 0.5 \\
  x_i, & \text{otherwise} 
\end{cases} \quad r = U(0, 1)
\]

Non-uniform Mutation:

- mutation probability sinks with time
- 'nonUnifMutation', 'multiNonUnifMutation'
Crossover Operators

Simple Crossover: 'simpleXover' (binary & real values)

Arithmetic Crossover: 'arithXover'

\[
\begin{align*}
X' &= r \bar{X} + (1 - r) \bar{Y} \quad r = U(0, 1) \\
\bar{Y}' &= (1 - r) \bar{X} + r \bar{Y}
\end{align*}
\]

Heuristic Crossover: 'heuristicXover'

\[
\begin{align*}
\bar{X}' &= \bar{X} + r(\bar{X} - \bar{Y}) \quad r = U(0, 1) \\
\bar{Y}' &= \bar{X}
\end{align*}
\]

if $X'$ is feasible (i.e. within bounds)
## Summary

### Selection

<table>
<thead>
<tr>
<th>Name</th>
<th>File</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roulette Wheel</td>
<td>roulette.m</td>
<td>None</td>
</tr>
<tr>
<td>Normalized Geometric Select</td>
<td>normGeomSelect.m</td>
<td>Probability of Selecting Best</td>
</tr>
<tr>
<td>Tournament</td>
<td>tourn.m</td>
<td>Number of individuals in each tournament</td>
</tr>
</tbody>
</table>

### Reproduction

<table>
<thead>
<tr>
<th>Name</th>
<th>File</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Crossover</td>
<td>arithXover.m</td>
<td>none</td>
</tr>
<tr>
<td>Heuristic Crossover</td>
<td>heuristicXover.m</td>
<td>number of retries (t)</td>
</tr>
<tr>
<td>Simple Crossover</td>
<td>simpleXover.m</td>
<td>none</td>
</tr>
<tr>
<td>Boundary Mutation</td>
<td>boundary.m</td>
<td>none</td>
</tr>
<tr>
<td>Multi-Non-Uniform Mutation</td>
<td>multiNonUnifMut.m</td>
<td>max num of generations, shape parameter (b)</td>
</tr>
<tr>
<td>Non-Uniform Mutation</td>
<td>nonUnifMut.m</td>
<td>max num of generations, shape parameter (b)</td>
</tr>
<tr>
<td>Uniform Mutation</td>
<td>unifMut.m</td>
<td>none</td>
</tr>
</tbody>
</table>
Termination cond. & initial population

'\texttt{maxGenTerm}': Terminate after fixed number of generations

'\texttt{optMaxGenTerm}': Terminate as soon as some optimal fitness value is reached

\[ \texttt{[pop]} = \texttt{initializega}(\texttt{popSize}, \texttt{bounds}, \texttt{evalFN}, \texttt{options}) \]

- \texttt{popSize} ... size of the population
- \texttt{bounds} ... a matrix which contains the bounds of each variable, i.e.
  \[ [\texttt{var1\_high var1\_low}; \texttt{var2\_high var2\_low}; \ldots] \]
- \texttt{evalFN} ... Name of evaluation Function
- \texttt{Options} ... [\texttt{epsilon floatGA display prec}]
Particle Swarm Optimization

Introduced by Russel Ebenhart (an Electrical Engineer) and James Kennedy (a Social Psychologist) in 1995

Belongs to the categories of Swarm Intelligence techniques and Evolutionary Algorithms for optimization.

Inspired by the social behavior of birds studied in late 80s and early 90s.

The particles in the swarm *co-operate*.
Basic idea

The velocity of a particle is calculated in dependence of previous best solutions:

Position of particle $i$

$\vec{p}_i$

Position with best performance of particle $i$ until now

$\vec{g}$

Position with best performance of all particles until now

$\vec{x}_i$

Previous velocity of particle $i$

$\vec{v}_i$

Current velocity of particle $i$
PSO algorithm

Pick two random numbers: $r_p, r_g \sim U(0, 1)$
Update the particle’s velocity $\vec{v}$ as follows:

$$
\vec{v} \leftarrow \omega \vec{v} + \phi_p r_p (\vec{p} - \vec{x}) + \phi_g r_g (\vec{g} - \vec{x})
$$

where $\omega$, $\phi_p$, and $\phi_g$ are user-defined behavioural parameters.

$\vec{p}$ particle’s best known position
$\vec{g}$ swarm’s best known position

See pso-pseudocode.pdf for help.
psom.m

\[ x = \text{pso}(\text{xbounds}, \text{npart}, \text{errorfct}, \text{maxiter}, \text{maxconsfail}, \text{params}); \]

\( x \) \hspace{1cm} ... \text{best solution found}

\( \text{Xbounds} \) \hspace{1cm} ... \text{D times 2 matrix containing two vectors, each consisting of D elements, that define upper and lower bounds on the search space of } x.\)

\( \text{npart} \) \hspace{1cm} ... \text{number of particles}

\( \text{errorfct} \) \hspace{1cm} ... \text{handle of the error function to minimize}

\( \text{maxiter} \) \hspace{1cm} ... \text{number of maximum iterations before termination}

\( \text{maxconsfail} \) \hspace{1cm} ... \text{number of maximum consecutive iterations without improvement of } x_{\text{best}}\)

\( \text{params} \) \hspace{1cm} ... \text{parameter vector } [w \ \phi_p \ \phi_g]
Simulated annealing

Simulated annealing (SA) is a probabilistic algorithm for locating a good approximation to the global optimum of a given function in a large search space.

**Basic idea**
Error function $f$ maps solutions $x$ onto errors in $\mathbb{R}$.

1. Start with a random solution $i$ (which is arbitrarily bad).
2. Choose some "nearby" solution $j$.
3. If the new solution is better (i.e. $f(\vec{x}_i) > f(\vec{x}_j)$), take it as the current solution (= accept it).
4. If it is worse, accept it with a probability that depends on $f(\vec{x}_i) - f(\vec{x}_j)$ and a global parameter $T$ (the temperature).
Convergence of SA

Move accepted with probability

Unconditional Acceptance

\[ \text{Move accepted with probability } = e^{-\frac{\Delta C}{T}} \]
Generation and acceptance

**Generation mechanism:** is a means of selecting a solution \( j \) from the neighborhood of solution \( i \).

\[ \vec{x}_j = \vec{x}_i + const \cdot N(0,1) \]

**Acceptance criterion:** defines whether \( \vec{x}_j \) is accepted from \( i \) by applying the following acceptance probability:

\[
P_T(\text{accept } \vec{x}_j) = \begin{cases} 
1 & \text{if } f(\vec{x}_j) \leq f(\vec{x}_i) \\
\exp\left(\frac{f(\vec{x}_i) - f(\vec{x}_j)}{T}\right) & \text{if } f(\vec{x}_j) > f(\vec{x}_i)
\end{cases}
\]
Asymptotic convergence

After sufficiently many transitions, we have

\[
P_T(\tilde{x} = \tilde{x}_i) = \frac{1}{N_0(T)} \exp \left( \frac{f(\tilde{x}_i)}{T} \right), \quad \text{with}
\]

\[
N_0(T) = \sum_j \exp \left( \frac{f(\tilde{x}_j)}{T} \right).
\]

This is the **stationary distribution** (equilibrium distribution).

**Asymptotic convergence:**

\[
\lim_{T \to 0} P(\tilde{x} \in X_{opt}) = \frac{1}{|X_{opt}|}
\]

\[
X_{opt} \ldots \text{best solutions}
\]
Annealing Schedule

Sequences of homogeneous Markov chains (one at each $T$):

- $T_{t+1} = \alpha T_t \quad \alpha < 1$ usually successful between 0.8 - 0.99

- $T_{t+1} = \frac{T_t}{1 + \beta T_t}$ Lundy & Mees

- $T_{t+1} = \frac{T_t}{1 + \frac{T_t \ln(1 + \Delta E)}{3\sigma T}}$ Aarts & Korst
anneal.m

[x, stat] = anneal(errorfunction, xinit, options);

errorfunction  ... handle of the cost function
xinit          ... initial parameters
options       ... option structure

options.Verbosity = 0;
options.CoolSched = @(T)(0.8*T);
options.Generator = @(x)(x(:)'+ ...)';
options.StopTemp = ...;
options.MaxIter = ...;
Assignment 2

2 Cart-Pole Controller Optimization [5 P]

Optimize the weights of a neural network that controls the horizontal forces applied to a cart in order to swing up a pole that is mounted on it into the upright position. Use an optimization algorithm of your choice (one of the four algorithms used in homework assignment 1).

\[(x_c, \dot{x}_c, \varphi, \dot{\varphi})^T\]

\(x_c\) is the position of the cart
\(\varphi\) is the pole angle
Neural Network Controller

Inputs

\[(xc, \dot{xc}, \sin \varphi, \cos \varphi, \dot{\varphi})^T\]

\(xc\) is the position of the cart
\(\varphi\) is the pole angle

\[w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, w_{o1}, w_{o2}\]

Output

The scalar output of the neural network, i.e. the force, should be bounded with values between -10 and 10.

Choose an appropriate number of hidden neurons

Hints:
Don’t forget to optimize the bias values of the neurons.
Use a tansig output neuron to obtain bounded output values.
Don’t use the MATLAB neural network toolbox.
Tasks

learn_cp.m
Modify this file that contains also the parameters for the cart-pole in the structure model (masses, length of the pole etc.) to implement an optimization algorithm that minimizes the error computed with the function cost_function.m.

cost_function.m
Modify this file that simulates the cart for a duration of 10 seconds and assigns an error value to the dynamics of the cart, e.g., score the cart-pole state at each simulation time step and sum up all these scores to obtain one final value.

Cart-pole state: \((xc, \dot{xc}, \varphi, \dot{\varphi})^T\)
- \(xc\) is the position of the cart
- \(\varphi\) is the pole angle
Assignment 3

3 Genetic Algorithm [3* P]

Apply genetic algorithms to an optimization problem of your choice for which you can show that the encoding of the variables (that should be optimized) in the genome has an effect on the convergence speed of the algorithm. Describe precisely the optimization problem and the key ideas behind the encoding scheme. Present your results clearly, structured and legible. Document them in such a way that anybody can reproduce them effortlessly.