Artificial Neural Network with Genetic Algorithm Research Report
By-Jun Yin

Summary about Pybrain:

Pybrain support GA, but not well supported,the reason are:

Supporting only one type of encoding : weight encoding.

Only support one point crossover.

Only support one type of generation change policy, which is fill population with child and parents

With minor bugs,those are:

In GA class,topProportion is set to 0.2, when having default population, which is 10, the parent is 2. with 2 parents, under the condition of one point crossover, could only produce 2 offsprings, is not enough to fill the rest of population, and rise an value error, fixed by change the topProportion to 0.5.

In GA class, the default value for elitism is False, the consequence is the eliteSize property is always return 0, In the produceOffspring method, es=min(self.eliteSize,self.selectionSize) always return 0. fixed by set elitism to True.

the topProportion=0.2,eliteProportion=0.5, it makes es=min(self.eliteSize,self.selectionSize) useless.

To get bet fitness,go to Source file ga.py, change the method produceOffspring like the following:

```
def produceOffspring(self):
    """ produce offspring by selection, mutation and crossover. """

    parents = self.select()
    #print parents
    es = min(self.eliteSize, self.selectionSize)
    #print es
    self.currentpop = parents[:es]
    '''Modified by JPQ '''
    nbchildren = self.populationSize - es
    if self.populationSize - es <= 0:
        nbchildren = len(parents)
    for child in self.crossOver(parents, nbchildren ):
        self.currentpop.append(self.mutated(child))
    print self.currentpop
    print "*"*10
    print self.fitnesses
    print "-"*60
    #
```

The code including two part:

```
from pybrain.datasets.classification import ClassificationDataSet
# below line can be replaced with the algorithm of choice e.g.
# from pybrain.optimization.hillclimber import HillClimber
from pybrain.optimization.populationbased.ga import GA
from pybrain.tools.shortcuts import buildNetwork

# create XOR dataset
d = ClassificationDataSet(2)
d.addSample([0., 0.], [0.])
d.addSample([0., 1.], [1.])
d.addSample([1., 0.], [1.])
d.addSample([1., 1.], [0.])
d.setField('class', [ [0.],[1.],[1.],[0.]])

nn = buildNetwork(2, 3, 1)
# d.evaluateModuleMSE takes nn as its first and only argument
ga = GA(d.evaluateModuleMSE, nn, minimize=True,storeAllPopulations=True,verbose=True,topProportion=0.5,elitism=True,maxLearningSteps=200)

ga.learn()
```

explaination:
d.evaluateModuleMSE is a fitness function
nn is the starting point-evaluable
topProportion=0.5 select 0.5 proportion as parents, take account their fitness,
maxLearningSteps=200, 200 generation
verbose=True, this help us find the individual which is fit most

run ga.learn() after 200 generation, this is the output:

```
[array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.03203013]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -5.4874984 ,  7.04407625,  6.30864974, -3.93880671,  1.55313018,
        1.4565474 ,  0.85763566,  3.03203013]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -5.4874984 ,  7.04407625,  6.30864974, -3.93880671,  1.55313018,
        1.4565474 ,  0.85763566,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.74921396,  0.89921799,
       -6.09348365,  7.04407625,  6.81854524, -4.08432261,  0.96775449,
        1.4565474 ,  0.85763566,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.03203013]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -5.4874984 ,  7.04407625,  6.30864974, -3.93880671,  1.55313018,
        1.31858099,  0.70649427,  3.01900042]), array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -5.4874984 ,  7.04407625,  6.30864974, -3.93880671,  1.55313018,
        1.4565474 ,  0.85763566,  3.01900042])]
**********
[-0.0018323549179017249, -0.0018362505575424811, -0.0019251665535430092, -0.0019267454027792814, -0.00196978214601
68798, -0.12395938596740852, -0.0020242725364468133, -0.022616389990873967, -0.0018362505575424811, -0.00196492596
11975271]
------------------------------------------------------------
Step: 200 best: -0.0018323549179
```

we can see in 200 generation the best is -0.0018323549179
using this find the index of the fitnesses list, which happen to be the first one,
the index is the same as the index of the evaluable list.
copy the evaluable, and paste into the next piece of code

```
from pybrain.utilities import setAllArgs, abstractMethod, DivergenceError
from pybrain.rl.learners.directsearch.directsearch import DirectSearchLearner
from pybrain.structure.parametercontainer import ParameterContainer
from pybrain.rl.environments.functions.function import FunctionEnvironment
from pybrain.rl.environments.fitnessevaluator import FitnessEvaluator
from pybrain.rl.environments.functions.transformations import oppositeFunction
from pybrain.structure.evolvables.maskedmodule import MaskedModule
from pybrain.structure.evolvables.maskedparameters import MaskedParameters
from pybrain.structure.evolvables.topology import TopologyEvolvable
from pybrain.structure.modules.module import Module

from pybrain.datasets.classification import ClassificationDataSet
# below line can be replaced with the algorithm of choice e.g.
# from pybrain.optimization.hillclimber import HillClimber
from pybrain.optimization.populationbased.ga import GA
from pybrain.tools.shortcuts import buildNetwork

from scipy import array, randn, ndarray, isinf, isnan, isscalar
import logging

evaluable=array([-0.83357998, -0.41181571, -3.13735002, -2.28029979,  0.89921799,
       -6.09348365,  7.04407625,  6.30864974, -4.08432261,  1.55313018,
        1.4565474 ,  0.85763566,  3.03203013])

nn = buildNetwork(2, 3, 1)
nn._setParameters(evaluable)

print nn.activate([0,0])
print nn.activate([0,1])
print nn.activate([1,0])
print nn.activate([1,1])
```

we have the weights that makes a network have most fitness, the next step is set those value to a network, using setParameters method.

then activate the work with XOR dataset: this is the output

```
[ 0.06376625]
[ 0.9795865]
[ 0.91468488]
[ 0.05382679]
```

now, the dataset is converged, and converged with genetic algorithm.

Next Step: integrate pybrain with ODE environment
Next Step: integrate pybrain,ODE environment,OpenGl

Thank you!

Reference:
http://stackoverflow.com/questions/15751723/how-to-train-a-neural-network-to-supervised-data-set-using-pybrain-black-box-opt/15868489#15868489
last updated May 18 2013
http://pybrain.org/docs/