ДОДАТОК А

Вихідний код програми для імітаційного моделювання

```
# Standard libraries
import numpy as np
import math
import copy
import json

# WANN functions
from domain import *  # Task environments
from .nsga_sort import nsga_sort
from .task import Task

from .ind import Ind


class Wann():
  """WANN main class. Evolves population given fitness
values of individuals.
  """
  def __init__(self, hyp):
    """Intialize WANN algorithm with hyperparameters
    Args:
      hyp - (dict) - algorithm hyperparameters

    Attributes:
      p       - (dict)     - algorithm hyperparameters
(see p/hypkey.txt)
      pop     - (Ind)      - Current population
      species - (Species)  - Current species
      innov   - (np_array) - innovation record
                [5 X nUniqueGenes]
                [0,:] == Innovation Number
                [1,:] == Source
                [2,:] == Destination
                [3,:] == New Node?
                [4,:] == Generation evolved
      gen     - (int)      - Current generation
    """
    self.p = hyp       # Hyperparameters
    self.pop = []      # Current population
    self.species = []  # Current species
    self.innov = []    # Innovation number (gene Id)
    self.gen = 0

  ''' Subfunctions '''
```

```
      from ._variation import evolvePop, recombine,
crossover,\
                              mutAddNode, mutAddConn,
topoMutate
      from ._speciate  import Species, speciate # Population
container


      def ask(self):
        """Returns newly evolved population
        """
        if len(self.pop) == 0:
          self.initPop()      # Initialize population
        else:
          self.probMoo()      # Rank population according to
objectives
          self.speciate()     # Divide population into species
          self.evolvePop()    # Create child population

        return self.pop       # Send child population for
evaluation


      def tell(self,reward):
        """Assigns fitness to current population

        Args:
          reward - (np_array) - fitness value of each
individual
                   [nInd X 1]

        """
        for i in range(np.shape(reward)[0]):
          self.pop[i].fitness = np.mean(reward[i,:])
          self.pop[i].fitMax  = np.max( reward[i,:])
          self.pop[i].nConn   = self.pop[i].nConn


      def initPop(self):
        """Initialize population with a list of random
individuals
        """
        ##  Create base individual
        p = self.p # readability

        # - Create Nodes -
        nodeId = np.arange(0,p['ann_nInput']+ p['ann_nOutput']
+1,1)
```

```
        node = np.empty((3,len(nodeId)))
        node[0,:] = nodeId

        # Node types: [1:input, 2:hidden, 3:bias, 4:output]
        node[1,0]             = 4 # Bias
        node[1,1:p['ann_nInput']+1] = 1 # Input Nodes
        node[1,(p['ann_nInput']+1):\
               (p['ann_nInput']+p['ann_nOutput']+1)]  = 2 #
Output Nodes

        # Node Activations
        node[2,:] = p['ann_initAct']

        # - Create Conns -
        nConn = (p['ann_nInput']+1) * p['ann_nOutput']
        ins   = np.arange(0,p['ann_nInput']+1,1)            #
Input and Bias Ids
        outs  = (p['ann_nInput']+1) +
np.arange(0,p['ann_nOutput']) # Output Ids

        conn = np.empty((5,nConn,))
        conn[0,:] = np.arange(0,nConn,1)      # Connection Id
        conn[1,:] = np.tile(ins, len(outs))   # Source Nodes
        conn[2,:] = np.repeat(outs,len(ins) ) # Destination
Nodes
        conn[3,:] = np.nan                    # Weight Values
        conn[4,:] = 1                         # Enabled?

        # Create population of individuals (for WANN weight
value doesn't matter)
        pop = []
        for i in range(p['popSize']):
            newInd = Ind(conn, node)
            newInd.conn[3,:] = 1 #(2*(np.random.rand(1,nConn)-
0.5))*p['ann_absWCap']
            newInd.conn[4,:] = np.random.rand(1,nConn) <
p['prob_initEnable']
            newInd.express()
            newInd.birth = 0
            pop.append(copy.deepcopy(newInd))

        # - Create Innovation Record -
        innov = np.zeros([5,nConn])
        innov[0:3,:] = pop[0].conn[0:3,:]
        innov[3,:] = -1

        self.pop = pop
       self.innov = innov
```

```
      def probMoo(self):
        """Rank population according to Pareto dominance.
        """
        # Compile objectives
        meanFit = np.asarray([ind.fitness for ind in
self.pop])
        maxFit  = np.asarray([ind.fitMax  for ind in
self.pop])
        nConns  = np.asarray([ind.nConn   for ind in
self.pop])
        nConns[nConns==0] = 1 # No conns is always pareto
optimal (but boring)
        objVals = np.c_[meanFit,maxFit,1/nConns] # Maximize

        # Alternate second objective
        if self.p['alg_probMoo'] < np.random.rand():
          rank = nsga_sort(objVals[:,[0,1]])
        else:
          rank = nsga_sort(objVals[:,[0,2]])

        # Assign ranks
        for i in range(len(self.pop)):
          self.pop[i].rank = rank[i]


    # -- Hyperparameter plumbing
------------------------------------------- -- #
    def loadHyp(pFileName, printHyp=False):
      """Loads hyperparameters from .json file
      Args:
          pFileName - (string) - file name of hyperparameter
file
          printHyp  - (bool)   - print contents of
hyperparameter file to terminal?

      Note: see p/hypkey.txt for detailed hyperparameter
description
      """

      # Load Parameters from disk
      with open(pFileName) as data_file:
        hyp = json.load(data_file)

      # Task hyper parameters
      task = Task(games[hyp['task']],paramOnly=True)
      hyp['ann_nInput']   = task.nInput
```

```
        hyp['ann_nOutput']  = task.nOutput
        hyp['ann_initAct']  = task.activations[0]
        hyp['ann_actRange'] = task.actRange

        if 'alg_act' in hyp:
          hyp['ann_actRange'] =
np.full_like(task.actRange,hyp['alg_act'])

        if printHyp is True:
          print(json.dumps(hyp, indent=4, sort_keys=True))
        return hyp

      def updateHyp(hyp,pFileName=None):
        """Overwrites default hyperparameters with those from
second .json file
        """
        print('\t*** Running with hyperparameters: ', pFileName,
'\t***')
        ''' Overwrites selected parameters those from file '''
        if pFileName != None:
          with open(pFileName) as data_file:
            update = json.load(data_file)

          hyp.update(update)

          # Task hyper parameters
          task = Task(games[hyp['task']],paramOnly=True)
          hyp['ann_nInput']   = task.nInput
          hyp['ann_nOutput']  = task.nOutput
          hyp['ann_initAct']  = task.activations[0]
          hyp['ann_actRange'] = task.actRange
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	_86_ с.
	Інші документи	
2.	Презентаційні матеріали	_20_ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Пуйда Б.Д.			Дослідження властивостей нейронних мереж з випадковою ініціалізацією
Перевірив		Узлов Д.Ю.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-20-1	122
ХНУРЕ кафедра ШІ	