[Dacon] 블럭 장난감 제조 공정 최적화 경진대회

_ (팀명)

2020년 월 일 (제출날짜)

1. 본 코드는 대회 참가를 돕고자 단순 예시를 작성한 것으로 참고용으로 사용바랍니다.
2. 본 코드는 자유롭게 수정하여 사용 할 수 있습니다.
3. 추가 모듈 보러가기: https://bit.ly/36MNs76 (https://bit.ly/36MNs76)

1. 라이브러리 및 데이터

Library & Data

In [1]:

```
import pandas as pd
import numpy as np
import multiprocessing
import warnings
from copy import deepcopy
from module.genome import Genome, genome_score
import datetime
warnings.filterwarnings(action='ignore')
np.random.seed(777)
```

In [2]:

```
!python --version
print('Pandas : %s'%(pd.__version__))
print('Numpy : %s'%(np.__version__))
```

```
Pandas : 1.0.4
Numpy : 1.18.1

Python 3.6.10 :: Anaconda, Inc.
```

2. 데이터 전처리

Data Cleansing & Pre-Processing

In [3]:

```
# 입력하세요.
```

3. 탐색적 자료분석

Exploratory Data Analysis

In [4]:

```
# 입력하세요.
```

4. 변수 선택 및 모델 구축

Feature Engineering & Initial Modeling

In [5]:

```
CPU_CORE = multiprocessing.cpu_count() # 멀티프로세싱 CPU 사용 수
N_POPULATION = 200                      # 세대당 생성수
N_BEST = 10                             # 베스트 수
N_CHILDREN = 5                         # 자손 유전자 수
PROB_MUTATION = 0.3                    # 돌연변이
REVERSE = True                         # 배열 순서 (False: ascending order, True: descending order)

score_ini = 10                         # 초기 점수
input_length = 125                     # 입력 데이터 길이
output_length_1 = 5 * 2                # Event (CHECK_1~4, PROCESS)
output_length_2 = 12 * 2                   # MOL(0~5.5, step:0.5)
h1 = 50                                # 히든레이어1 노드 수
h2 = 50                                # 히든레이어2 노드 수
h3 = 50                                # 히든레이어3 노드 수
EPOCHS = 2000                            # 반복 횟수

genomes = []
for _ in range(N_POPULATION):
    genome = Genome(score_ini, input_length, output_length_1, output_length_2, h1, h2, h3)
    genomes.append(genome)
try:
    for i in range(N_BEST):
        genomes[i] = best_genomes[i]
except:
    best_genomes = []
    for _ in range(N_BEST):
        genome = Genome(score_ini, input_length, output_length_1, output_length_2, h1, h2, h3)
        best_genomes.append(genome)
```

In [6]:

```
best_genomes[0].forward(np.zeros((1, 125)))
```

Out[6]:

```
('CHECK_1', 'CHECK_1', 0.0, 0.0)
```

5. 모델 학습 및 검증

Model Tuning & Evaluation

1. PRT는 고정값 사용
2. Event A, Event B (MOL_A, MOL_B) 를 같은 값으로 제한
3. Event는 CHECK와 PROCESS 만 사용함

4. 목적 함수로 수요 부족분만 고려함
5. Event와 MOL에 대해 인공신경망 모델을 만들어 유전 알고리즘으로 학습

In [7]:

```
n_gen = 1
score_history = []
high_score_history = []
mean_score_history = []
best_gen = None
best_score_ever = 0
while n_gen <= EPOCHS:
    print('EPOCH', n_gen, datetime.datetime.now())
    genomes = np.array(genomes)
    while len(genomes)%CPU_CORE != 0:
        genomes = np.append(genomes, Genome(score_ini, input_length, output_length_1, output_length_
    genomes = genomes.reshape((len(genomes)//CPU_CORE, CPU_CORE))

    for idx, _genomes in enumerate(genomes):
        if __name__ == '__main__':
            pool = multiprocessing.Pool(processes=CPU_CORE)
            genomes[idx] = pool.map(genome_score, _genomes)
            pool.close()
            pool.join()
    genomes = list(genomes.reshape(genomes.shape[0]*genomes.shape[1]))

     # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    # 평균 점수
    s = 0
    for i in range(N_BEST):
        s += genomes[i].score
    s /= N_BEST

    # Best Score
    bs = genomes[0].score

    # Best Model 추가
    if best_genomes is not None:
        genomes.extend(best_genomes)

    # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    score_history.append([n_gen, genomes[0].score])
    high_score_history.append([n_gen, bs])
    mean_score_history.append([n_gen, s])

    if genomes[0].score > best_score_ever:
        best_score_ever = genomes[0].score
        best_gen = genomes[0]

    # 결과 출력
    print('EPOCH #%s₩tHistory Best Score: %s₩tBest Score: %s₩tMean Score: %s' % (n_gen, genomes[0].s

    # 모델 업데이트
    best_genomes = deepcopy(genomes[:N_BEST])

    # CHILDREN 생성
    for i in range(N_CHILDREN):
        new_genome = deepcopy(best_genomes[0])
        a_genome = np.random.choice(best_genomes)
        b_genome = np.random.choice(best_genomes)
        for j in range(input_length):
            cut = np.random.randint(new_genome.w1.shape[1])
            new_genome.w1[j, :cut] = a_genome.w1[j, :cut]
            new_genome.w1[j, cut:] = b_genome.w1[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w2.shape[1])
            new_genome.w2[j, :cut] = a_genome.w2[j, :cut]
            new_genome.w2[j, cut:] = b_genome.w2[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w3.shape[1])
            new_genome.w3[j, :cut] = a_genome.w3[j, :cut]
            new_genome.w3[j, cut:] = b_genome.w3[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w4.shape[1])
            new_genome.w4[j, :cut] = a_genome.w4[j, :cut]
            new_genome.w4[j, cut:] = b_genome.w4[j, cut:]

        for j in range(input_length):
            cut = np.random.randint(new_genome.w5.shape[1])
            new_genome.w5[j, :cut] = a_genome.w5[j, :cut]
            new_genome.w5[j, cut:] = b_genome.w5[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w6.shape[1])
            new_genome.w6[j, :cut] = a_genome.w6[j, :cut]
            new_genome.w6[j, cut:] = b_genome.w6[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w7.shape[1])
            new_genome.w7[j, :cut] = a_genome.w7[j, :cut]
            new_genome.w7[j, cut:] = b_genome.w7[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w8.shape[1])
            new_genome.w8[j, :cut] = a_genome.w8[j, :cut]
            new_genome.w8[j, cut:] = b_genome.w8[j, cut:]

        best_genomes.append(new_genome)

    # 모델 초기화
    genomes = []
    for i in range(int(N_POPULATION / len(best_genomes))):
        for bg in best_genomes:
            new_genome = deepcopy(bg)
            mean = 0
            stddev = 0.2
            # 50% 확률로 모델 변형
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w1 += new_genome.w1 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w2 += new_genome.w2 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w3 += new_genome.w3 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w4 += new_genome.w4 * np.random.normal(mean, stddev, size=(h3, output_le
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w5 += new_genome.w5 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w6 += new_genome.w6 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w7 += new_genome.w7 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w8 += new_genome.w8 * np.random.normal(mean, stddev, size=(h3, output_le
            genomes.append(new_genome)

    if REVERSE:
        if bs < score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_le
    else:
        if bs > score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_le

    n_gen += 1
```

```
EPOCH 1 2020-07-01 01:19:28.931909
EPOCH #1        History Best Score: 82.6869582127399    Best Score: 82.6869582127399
Mean Score: 80.03549654494219
EPOCH 2 2020-07-01 01:23:06.120624
EPOCH #2        History Best Score: 83.60988521644455   Best Score: 83.6098852164445
5       Mean Score: 82.8663116117162
EPOCH 3 2020-07-01 01:26:36.165054
EPOCH #3        History Best Score: 84.50513458489384   Best Score: 84.5051345848938
4       Mean Score: 83.53301577870671
EPOCH 4 2020-07-01 01:30:04.535739
EPOCH #4        History Best Score: 84.50513458489384   Best Score: 84.2918558451505
Mean Score: 84.00719192106621
EPOCH 5 2020-07-01 01:33:33.706981
EPOCH #5        History Best Score: 84.50513458489384   Best Score: 84.3723398143764
6       Mean Score: 84.21894154465492
EPOCH 6 2020-07-01 01:37:04.920473
EPOCH #6        History Best Score: 85.2736793501118    Best Score: 85.2736793501118
Mean Score: 84.50485883155666
EPOCH 7 2020-07-01 01:40:33.005484
EPOCH #7        History Best Score: 85.2736793501118    Best Score: 85.2736793501118
Mean Score: 84.88010321088586
EPOCH 8 2020-07-01 01:43:57.567251
EPOCH #8        History Best Score: 85.2736793501118    Best Score: 85.2736793501118
Mean Score: 85.10633314112114
EPOCH 9 2020-07-01 01:47:30.185023
EPOCH #9        History Best Score: 85.47250039399918   Best Score: 85.4725003939991
8       Mean Score: 85.29356145450055
EPOCH 10 2020-07-01 01:50:59.566643
EPOCH #10       History Best Score: 85.55640823905259   Best Score: 85.5564082390525
9       Mean Score: 85.33330787571316
EPOCH 11 2020-07-01 01:56:13.120743
EPOCH #11       History Best Score: 85.55640823905259   Best Score: 85.5218395352745
2       Mean Score: 85.38879251312486
EPOCH 12 2020-07-01 02:05:38.991357
EPOCH #12       History Best Score: 85.89173179301419   Best Score: 85.8917317930141
9       Mean Score: 85.5402344579232
EPOCH 13 2020-07-01 02:15:44.169029
EPOCH #13       History Best Score: 85.89173179301419   Best Score: 85.8917317930141
9       Mean Score: 85.45034667318775
EPOCH 14 2020-07-01 02:25:26.713636
EPOCH #14       History Best Score: 85.89173179301419   Best Score: 85.8917317930141
9       Mean Score: 85.68047693407509
EPOCH 15 2020-07-01 02:34:16.314313
EPOCH #15       History Best Score: 86.13121078633017   Best Score: 86.1312107863301
7       Mean Score: 85.7734332119949
EPOCH 16 2020-07-01 02:42:38.291712
EPOCH #16       History Best Score: 86.13121078633017   Best Score: 86.0730405338358
5       Mean Score: 85.89640325607672
EPOCH 17 2020-07-01 02:51:19.630588
EPOCH #17       History Best Score: 86.76362753269271   Best Score: 86.7636275326927
1       Mean Score: 86.0217020887025
EPOCH 18 2020-07-01 03:00:12.548331
EPOCH #18       History Best Score: 86.76362753269271   Best Score: 86.7123375963070
7       Mean Score: 85.91708758271295
EPOCH 19 2020-07-01 03:09:24.764920
EPOCH #19       History Best Score: 86.76362753269271   Best Score: 86.7123375963070
7       Mean Score: 86.22366219233119
EPOCH 20 2020-07-01 03:19:10.406509
EPOCH #20       History Best Score: 87.312934570332     Best Score: 87.312934570332
Mean Score: 85.47665291484118
EPOCH 21 2020-07-01 03:28:47.354493
EPOCH #21       History Best Score: 87.312934570332     Best Score: 86.7636275326927
1       Mean Score: 86.10865722932411
EPOCH 22 2020-07-01 03:39:12.066736
EPOCH #22       History Best Score: 87.312934570332     Best Score: 86.8059124759645
2       Mean Score: 86.58198807833779
EPOCH 23 2020-07-01 03:48:56.528626
EPOCH #23       History Best Score: 87.312934570332     Best Score: 86.9023011087582
2       Mean Score: 86.10724642044457
EPOCH 24 2020-07-01 03:59:02.432171
EPOCH #24       History Best Score: 87.35614422120726   Best Score: 87.3561442212072
6       Mean Score: 86.72619370996071
EPOCH 25 2020-07-01 04:09:12.563410
EPOCH #25       History Best Score: 87.40199773707766   Best Score: 87.4019977370776
6       Mean Score: 87.00281387602098
EPOCH 26 2020-07-01 04:19:08.761911
EPOCH #26       History Best Score: 87.40199773707766   Best Score: 87.3561442212072
6       Mean Score: 87.08405767226213
EPOCH 27 2020-07-01 04:28:20.395599
EPOCH #27       History Best Score: 87.40199773707766   Best Score: 87.4019977370776
6       Mean Score: 87.36531492438135
EPOCH 28 2020-07-01 04:37:58.458603
EPOCH #28       History Best Score: 87.44379882171975   Best Score: 87.4437988217197
5       Mean Score: 86.38105870111455
EPOCH 29 2020-07-01 04:47:38.625978
EPOCH #29       History Best Score: 87.44379882171975   Best Score: 87.4437988217197
5       Mean Score: 87.16581122509011
EPOCH 30 2020-07-01 04:57:05.065068
EPOCH #30       History Best Score: 87.44379882171975   Best Score: 87.4437988217197
5       Mean Score: 87.36219524970637
EPOCH 31 2020-07-01 05:07:20.715547
EPOCH #31       History Best Score: 87.67033375185825   Best Score: 87.6703337518582
5       Mean Score: 87.41940054984772
EPOCH 32 2020-07-01 05:18:02.167020
EPOCH #32       History Best Score: 87.67033375185825   Best Score: 87.6703337518582
5       Mean Score: 87.4076450277919
EPOCH 33 2020-07-01 05:27:36.390294
EPOCH #33       History Best Score: 87.69488151300597   Best Score: 87.6948815130059
7       Mean Score: 87.50670130377627
EPOCH 34 2020-07-01 05:37:40.510609
EPOCH #34       History Best Score: 87.69488151300597   Best Score: 87.5901668252391
8       Mean Score: 87.48443913718943
EPOCH 35 2020-07-01 05:46:45.806247
EPOCH #35       History Best Score: 87.87384561315334   Best Score: 87.8738456131533
```

```
4       Mean Score: 87.61735697676525
EPOCH 36 2020-07-01 05:56:51.631003
EPOCH #36       History Best Score: 88.20019612403124   Best Score: 88.2001961240312
4       Mean Score: 87.59825715657635
EPOCH 37 2020-07-01 06:06:46.638689
EPOCH #37       History Best Score: 88.20019612403124   Best Score: 87.8418706988424
Mean Score: 87.52804843993223
EPOCH 38 2020-07-01 06:17:01.481328
EPOCH #38       History Best Score: 88.20019612403124   Best Score: 87.9760503802821
6       Mean Score: 87.7162929793188
EPOCH 39 2020-07-01 06:27:43.558130
EPOCH #39       History Best Score: 88.20019612403124   Best Score: 88.2001961240312
4       Mean Score: 87.7667351953852
EPOCH 40 2020-07-01 06:37:44.045290
EPOCH #40       History Best Score: 88.20019612403124   Best Score: 88.2001961240312
4       Mean Score: 87.86035769287602
EPOCH 41 2020-07-01 06:47:19.048478
EPOCH #41       History Best Score: 88.32168150551      Best Score: 88.32168150551
Mean Score: 87.91628013786436
EPOCH 42 2020-07-01 06:56:59.419308
EPOCH #42       History Best Score: 88.32168150551      Best Score: 88.32168150551
Mean Score: 88.21536673565619
EPOCH 43 2020-07-01 07:07:06.545585
```

```
---------------------------------------------------------------------------
----
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-7-31ece2cd64e8> in <module>
     15         if __name__ == '__main__':
     16             pool = multiprocessing.Pool(processes=CPU_CORE)
---> 17             genomes[idx] = pool.map(genome_score, _genomes)
     18             pool.close()
     19             pool.join()

~₩anaconda3₩envs₩tensorflow2.0₩lib₩multiprocessing₩pool.py in map(self,
 func, iterable, chunksize)
    264         in a list that is returned.
    265         '''
--> 266         return self._map_async(func, iterable, mapstar, chunksize).ge
t()
    267 
    268     def starmap(self, func, iterable, chunksize=None):

~₩anaconda3₩envs₩tensorflow2.0₩lib₩multiprocessing₩pool.py in get(self,
 timeout)
    636 
    637     def get(self, timeout=None):
--> 638         self.wait(timeout)
    639         if not self.ready():
    640             raise TimeoutError

~₩anaconda3₩envs₩tensorflow2.0₩lib₩multiprocessing₩pool.py in wait(self,
timeout)
    633 
    634     def wait(self, timeout=None):
--> 635         self._event.wait(timeout)
    636 
    637     def get(self, timeout=None):

~₩anaconda3₩envs₩tensorflow2.0₩lib₩threading.py in wait(self, timeout)
    549             signaled = self._flag
    550             if not signaled:
--> 551                 signaled = self._cond.wait(timeout)
    552             return signaled
    553 

~₩anaconda3₩envs₩tensorflow2.0₩lib₩threading.py in wait(self, timeout)
    293         try:    # restore state no matter what (e.g., KeyboardInt
errupt)
    294             if timeout is None:
--> 295                 waiter.acquire()
    296                 gotit = True
    297             else:

KeyboardInterrupt: 
```

6. 결과 및 결언

Conclusion & Discussion

결과 그래프

In [ ]:

```
import matplotlib.pyplot as plt

# Score Graph
score_history = np.array(score_history)
high_score_history = np.array(high_score_history)
mean_score_history = np.array(mean_score_history)

plt.plot(score_history[:,0], score_history[:,1], '-o', label='BEST')
plt.plot(high_score_history[:,0], high_score_history[:,1], '-o', label='High')
plt.plot(mean_score_history[:,0], mean_score_history[:,1], '-o', label='Mean')
plt.legend()
plt.xlim(0, EPOCHS)
plt.ylim(bottom=0)
plt.xlabel('Epochs')
plt.ylabel('Score')
plt.show()
```

Submission 파일 만들기

In [8]:

```
# 재고 계산
from module.simulator import Simulator
simulator = Simulator()
order = pd.read_csv('module/order.csv')
submission = best_gen.predict(order)
_, df_stock = simulator.get_score(submission)

# PRT 개수 계산
PRTs = df_stock[['PRT_1', 'PRT_2', 'PRT_3', 'PRT_4']].values
PRTs = (PRTs[:-1] - PRTs[1:])[24*23:]
PRTs = np.ceil(PRTs * 1.1)
PAD = np.zeros((24*23+1, 4))
PRTs = np.append(PRTs, PAD, axis=0).astype(int)

# Submission 파일에 PRT 입력
submission.loc[:, 'PRT_1':'PRT_4'] = PRTs
submission.to_csv('Dacon_baseline2.csv', index=False)
```

점수 향상 팁

해당 코드는 단순한 모델로 다음 방법으로 점수 향상을 꾀할 수 있습니다.

1. 성형 공정 2개 라인을 따로 모델링
2. CHANGE, STOP 이벤트 활용
3. 수요 초과분 외 다양한 양상을 반영하는 목적함수
4. 유전 알고리즘 외 효율적인 학습 기법

In [ ]:

```
while n_gen <= EPOCHS:
    print('EPOCH', n_gen, datetime.datetime.now())
    genomes = np.array(genomes)
    while len(genomes)%CPU_CORE != 0:
        genomes = np.append(genomes, Genome(score_ini, input_length, output_length_1, output_length_
    genomes = genomes.reshape((len(genomes)//CPU_CORE, CPU_CORE))

    for idx, _genomes in enumerate(genomes):
        if __name__ == '__main__':
            pool = multiprocessing.Pool(processes=CPU_CORE)
            genomes[idx] = pool.map(genome_score, _genomes)
            pool.close()
            pool.join()
    genomes = list(genomes.reshape(genomes.shape[0]*genomes.shape[1]))

     # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    # 평균 점수
    s = 0
    for i in range(N_BEST):
        s += genomes[i].score
    s /= N_BEST

    # Best Score
    bs = genomes[0].score

    # Best Model 추가
    if best_genomes is not None:
        genomes.extend(best_genomes)

    # score에 따라 정렬
    genomes.sort(key=lambda x: x.score, reverse=REVERSE)

    score_history.append([n_gen, genomes[0].score])
    high_score_history.append([n_gen, bs])
    mean_score_history.append([n_gen, s])

    if genomes[0].score > best_score_ever:
        best_score_ever = genomes[0].score
        best_gen = genomes[0]

    # 결과 출력
    print('EPOCH #%s\tHistory Best Score: %s\tBest Score: %s\tMean Score: %s' % (n_gen, genomes[0].s

    # 모델 업데이트
    best_genomes = deepcopy(genomes[:N_BEST])

    # CHILDREN 생성
    for i in range(N_CHILDREN):
        new_genome = deepcopy(best_genomes[0])
        a_genome = np.random.choice(best_genomes)
        b_genome = np.random.choice(best_genomes)

        for j in range(input_length):
            cut = np.random.randint(new_genome.w1.shape[1])
            new_genome.w1[j, :cut] = a_genome.w1[j, :cut]
            new_genome.w1[j, cut:] = b_genome.w1[j, cut:]
        for j in range(h1):
            cut = np.random.randint(new_genome.w2.shape[1])
            new_genome.w2[j, :cut] = a_genome.w2[j, :cut]
            new_genome.w2[j, cut:] = b_genome.w2[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w3.shape[1])
            new_genome.w3[j, :cut] = a_genome.w3[j, :cut]
            new_genome.w3[j, cut:] = b_genome.w3[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w4.shape[1])
            new_genome.w4[j, :cut] = a_genome.w4[j, :cut]
            new_genome.w4[j, cut:] = b_genome.w4[j, cut:]

        for j in range(input_length):
            cut = np.random.randint(new_genome.w5.shape[1])
            new_genome.w5[j, :cut] = a_genome.w5[j, :cut]
            new_genome.w5[j, cut:] = b_genome.w5[j, cut:]

        for j in range(h1):
            cut = np.random.randint(new_genome.w6.shape[1])
            new_genome.w6[j, :cut] = a_genome.w6[j, :cut]
            new_genome.w6[j, cut:] = b_genome.w6[j, cut:]

        for j in range(h2):
            cut = np.random.randint(new_genome.w7.shape[1])
            new_genome.w7[j, :cut] = a_genome.w7[j, :cut]
            new_genome.w7[j, cut:] = b_genome.w7[j, cut:]

        for j in range(h3):
            cut = np.random.randint(new_genome.w8.shape[1])
            new_genome.w8[j, :cut] = a_genome.w8[j, :cut]
            new_genome.w8[j, cut:] = b_genome.w8[j, cut:]

        best_genomes.append(new_genome)

    # 모델 초기화
    genomes = []
    for i in range(int(N_POPULATION / len(best_genomes))):
        for bg in best_genomes:
            new_genome = deepcopy(bg)
            mean = 0
            stddev = 0.2
            # 50% 확률로 모델 변형
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w1 += new_genome.w1 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w2 += new_genome.w2 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w3 += new_genome.w3 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w4 += new_genome.w4 * np.random.normal(mean, stddev, size=(h3, output_le
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w5 += new_genome.w5 * np.random.normal(mean, stddev, size=(input_length,
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w6 += new_genome.w6 * np.random.normal(mean, stddev, size=(h1, h2)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w7 += new_genome.w7 * np.random.normal(mean, stddev, size=(h2, h3)) * np
            if np.random.uniform(0, 1) < PROB_MUTATION:
                new_genome.w8 += new_genome.w8 * np.random.normal(mean, stddev, size=(h3, output_le
            genomes.append(new_genome)

    if REVERSE:
        if bs < score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_ler
    else:
        if bs > score_ini:
            genomes[len(genomes)//2:] = [Genome(score_ini, input_length, output_length_1, output_ler

    n_gen += 1
```