[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 [3]:

```
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 [4]:

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

```
Python 3.7.6
Pandas : 1.0.1
Numpy : 1.18.1
```

2. 데이터 전처리

Data Cleansing & Pre-Processing

In [4]:

```
# 입력하세요.
```

3. 탐색적 자료분석

Exploratory Data Analysis

In [5]:

```
# 입력하세요.
```

4. 변수 선택 및 모델 구축

Feature Engineering & Initial Modeling

In [6]:

```
CPU_CORE = multiprocessing.cpu_count() # 멀티프로세싱 CPU 사용 수
N_POPULATION = 300                     # 세대당 생성수
N_BEST = 20                            # 베스트 수
N_CHILDREN = 10                        # 자손 유전자 수
PROB_MUTATION = 0.4                    # 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 = 500                           # 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 [7]:

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

Out[7]:

```
('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 [8]:

```
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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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 2020-06-28 12:05:51.216003
EPOCH #1        History Best Score: 82.6869582127399    Best Score: 82.6869582127
399     Mean Score: 79.76876010534907
EPOCH 2 2020-06-28 12:10:06.630384
EPOCH #2        History Best Score: 83.83997335859024   Best Score: 83.8399733585
9024    Mean Score: 82.53936537102126
EPOCH 3 2020-06-28 12:14:44.404786
EPOCH #3        History Best Score: 86.00186999839943   Best Score: 86.0018699983
9943    Mean Score: 83.57946752817057
EPOCH 4 2020-06-28 12:19:43.802378
EPOCH #4        History Best Score: 86.00186999839943   Best Score: 85.7883814329
7729    Mean Score: 84.58583819521654
EPOCH 5 2020-06-28 12:24:52.194858
EPOCH #5        History Best Score: 86.00186999839943   Best Score: 86.0018699983
9943    Mean Score: 85.0754801183257
EPOCH 6 2020-06-28 12:30:53.014518
EPOCH #6        History Best Score: 86.00186999839943   Best Score: 85.6253917334
8269    Mean Score: 83.0644396164399
EPOCH 7 2020-06-28 12:36:35.425397
EPOCH #7        History Best Score: 86.00186999839943   Best Score: 86.0018699983
9943    Mean Score: 83.85238588386225
EPOCH 8 2020-06-28 12:41:22.206497
EPOCH #8        History Best Score: 86.00186999839943   Best Score: 86.0018699983
9943    Mean Score: 83.70574988410434
EPOCH 9 2020-06-28 12:45:48.513585
EPOCH #9        History Best Score: 86.00186999839943   Best Score: 86.0018699983
9943    Mean Score: 84.67572926711853
EPOCH 10 2020-06-28 12:50:24.679715
EPOCH #10       History Best Score: 86.00790855409831   Best Score: 86.0079085540
9831    Mean Score: 84.91629480722919
EPOCH 11 2020-06-28 12:54:54.801382
EPOCH #11       History Best Score: 86.00790855409831   Best Score: 85.6852467356
4391    Mean Score: 85.09180775113876
EPOCH 12 2020-06-28 12:59:30.960789
EPOCH #12       History Best Score: 86.12835740286806   Best Score: 86.1283574028
6806    Mean Score: 85.47453538051106
EPOCH 13 2020-06-28 13:04:13.369139
EPOCH #13       History Best Score: 86.12835740286806   Best Score: 86.0097155740
7291    Mean Score: 83.21133964940373
EPOCH 14 2020-06-28 13:08:55.401278
EPOCH #14       History Best Score: 86.25345928199066   Best Score: 86.2534592819
9066    Mean Score: 83.91787585754341
```

```
EPOCH 15 2020-06-28 13:13:16.180506
EPOCH #15       History Best Score: 86.25345928199066   Best Score: 85.7214882078
2758    Mean Score: 82.66563588393906
EPOCH 16 2020-06-28 13:17:18.392418
EPOCH #16       History Best Score: 86.50938117399771   Best Score: 86.5093811739
9771    Mean Score: 83.04059348102643
EPOCH 17 2020-06-28 13:21:19.634424
EPOCH #17       History Best Score: 86.88939083888184   Best Score: 86.8893908388
8184    Mean Score: 83.04901042461897
EPOCH 18 2020-06-28 13:25:21.274521
EPOCH #18       History Best Score: 87.27176081224349   Best Score: 87.2717608122
4349    Mean Score: 84.15547990634954
EPOCH 19 2020-06-28 13:29:22.491870
EPOCH #19       History Best Score: 87.27176081224349   Best Score: 87.1165020984
104     Mean Score: 85.59355882783942
EPOCH 20 2020-06-28 13:33:24.115290
EPOCH #20       History Best Score: 87.57459548650526   Best Score: 87.5745954865
0526    Mean Score: 86.07775649653173
EPOCH 21 2020-06-28 13:37:26.452395
EPOCH #21       History Best Score: 87.57459548650526   Best Score: 86.9329095448
5647    Mean Score: 84.15379670810238
EPOCH 22 2020-06-28 13:41:29.419829
EPOCH #22       History Best Score: 87.80434354515691   Best Score: 87.8043435451
5691    Mean Score: 84.9330629050012
EPOCH 23 2020-06-28 13:45:29.988073
EPOCH #23       History Best Score: 87.80434354515691   Best Score: 87.8043435451
5691    Mean Score: 84.66166916481085
EPOCH 24 2020-06-28 13:49:32.185294
EPOCH #24       History Best Score: 87.89620264675278   Best Score: 87.8962026467
5278    Mean Score: 85.99744120196013
EPOCH 25 2020-06-28 13:53:32.842477
EPOCH #25       History Best Score: 88.08642913252903   Best Score: 88.0864291325
2903    Mean Score: 85.29718314402429
EPOCH 26 2020-06-28 13:57:34.696960
EPOCH #26       History Best Score: 88.48808801495998   Best Score: 88.4880880149
5998    Mean Score: 85.58805838437915
EPOCH 27 2020-06-28 14:01:44.714118
EPOCH #27       History Best Score: 88.48808801495998   Best Score: 87.9539633858
7986    Mean Score: 84.51000154220338
EPOCH 28 2020-06-28 14:05:51.065762
EPOCH #28       History Best Score: 88.48808801495998   Best Score: 88.3142737066
9196    Mean Score: 83.4300515839381
EPOCH 29 2020-06-28 14:09:57.472962
EPOCH #29       History Best Score: 88.48808801495998   Best Score: 86.2606849041
9445    Mean Score: 84.09396053802622
EPOCH 30 2020-06-28 14:14:02.428981
EPOCH #30       History Best Score: 88.48808801495998   Best Score: 88.2970726092
0843    Mean Score: 82.87598518993218
EPOCH 31 2020-06-28 14:18:07.980000
EPOCH #31       History Best Score: 88.48808801495998   Best Score: 87.2855672190
2329    Mean Score: 83.43625036596609
EPOCH 32 2020-06-28 14:22:13.396555
EPOCH #32       History Best Score: 88.48808801495998   Best Score: 86.0468768384
7078    Mean Score: 82.82977246663141
EPOCH 33 2020-06-28 14:26:19.129013
EPOCH #33       History Best Score: 88.48808801495998   Best Score: 87.9127963864
5159    Mean Score: 84.17759037173553
EPOCH 34 2020-06-28 14:30:25.150382
EPOCH #34       History Best Score: 88.48808801495998   Best Score: 87.2020873200
4683    Mean Score: 82.65179533056452
EPOCH 35 2020-06-28 14:34:31.090190
```

```
EPOCH #35       History Best Score: 88.48808801495998   Best Score: 88.4880880149
5998    Mean Score: 84.49332531612863
EPOCH 36 2020-06-28 14:38:37.764235
EPOCH #36       History Best Score: 88.48808801495998   Best Score: 88.4880880149
5998    Mean Score: 85.16880541909377
EPOCH 37 2020-06-28 14:42:46.120017
EPOCH #37       History Best Score: 88.48808801495998   Best Score: 87.9015800825
9698    Mean Score: 85.69513926372997
EPOCH 38 2020-06-28 14:46:57.651219
EPOCH #38       History Best Score: 88.48808801495998   Best Score: 88.1910854500
4843    Mean Score: 85.86920748257418
EPOCH 39 2020-06-28 14:51:04.845301
EPOCH #39       History Best Score: 88.48808801495998   Best Score: 88.0350021796
6791    Mean Score: 85.44241160670865
EPOCH 40 2020-06-28 14:55:11.101013
EPOCH #40       History Best Score: 88.48808801495998   Best Score: 88.4880880149
5998    Mean Score: 86.23155682973368
EPOCH 41 2020-06-28 14:59:18.027502
EPOCH #41       History Best Score: 88.53444873755907   Best Score: 88.5344487375
5907    Mean Score: 85.5982314890281
EPOCH 42 2020-06-28 15:03:25.194588
EPOCH #42       History Best Score: 88.53444873755907   Best Score: 88.4880880149
5998    Mean Score: 86.97519242980901
EPOCH 43 2020-06-28 15:07:31.544116
EPOCH #43       History Best Score: 88.733825527644     Best Score: 88.7338255276
44      Mean Score: 87.59936405615463
EPOCH 44 2020-06-28 15:11:36.892880
EPOCH #44       History Best Score: 88.733825527644     Best Score: 88.4880880149
5998    Mean Score: 86.52789761428511
EPOCH 45 2020-06-28 15:15:41.606579
EPOCH #45       History Best Score: 88.733825527644     Best Score: 86.8024078034
262     Mean Score: 85.3459088293601
EPOCH 46 2020-06-28 15:19:46.931973
EPOCH #46       History Best Score: 88.7643955622637    Best Score: 88.7643955622
637     Mean Score: 85.06132327501327
EPOCH 47 2020-06-28 15:23:52.155377
EPOCH #47       History Best Score: 88.7643955622637    Best Score: 87.8388572580
177     Mean Score: 85.61836790336835
EPOCH 48 2020-06-28 15:27:57.849435
EPOCH #48       History Best Score: 88.7643955622637    Best Score: 88.2333673212
0195    Mean Score: 84.78868412879181
EPOCH 49 2020-06-28 15:32:03.787467
EPOCH #49       History Best Score: 88.7643955622637    Best Score: 88.6866273456
0071    Mean Score: 84.92360372992026
EPOCH 50 2020-06-28 15:36:08.823038
EPOCH #50       History Best Score: 88.7643955622637    Best Score: 88.7643955622
637     Mean Score: 85.03941033147234
EPOCH 51 2020-06-28 15:40:13.364482
EPOCH #51       History Best Score: 88.7643955622637    Best Score: 88.7308354349
1825    Mean Score: 85.1223510429688
EPOCH 52 2020-06-28 15:44:17.884383
EPOCH #52       History Best Score: 88.7643955622637    Best Score: 88.2085470631
6246    Mean Score: 84.75070912170288
EPOCH 53 2020-06-28 15:48:24.316822
EPOCH #53       History Best Score: 88.7643955622637    Best Score: 88.0350713023
2931    Mean Score: 84.17255770436577
EPOCH 54 2020-06-28 15:52:29.449371
EPOCH #54       History Best Score: 88.7643955622637    Best Score: 88.7643955622
637     Mean Score: 85.53179817534605
EPOCH 55 2020-06-28 15:56:34.072299
EPOCH #55       History Best Score: 88.7643955622637    Best Score: 88.3304133873
```

```
084     Mean Score: 85.3367471274926
EPOCH 56 2020-06-28 16:00:39.104144
EPOCH #56       History Best Score: 88.7643955622637    Best Score: 88.6354553785
5717    Mean Score: 86.39956093285485
EPOCH 57 2020-06-28 16:04:44.009651
EPOCH #57       History Best Score: 88.7643955622637    Best Score: 88.7643955622
637     Mean Score: 86.4990950768383
EPOCH 58 2020-06-28 16:08:49.115399
EPOCH #58       History Best Score: 88.7643955622637    Best Score: 88.6077617682
8671    Mean Score: 85.9056959069861

EPOCH 59 2020-06-28 16:12:53.994333
EPOCH #59       History Best Score: 88.7643955622637    Best Score: 88.5854262395
1524    Mean Score: 86.28655366084368
EPOCH 60 2020-06-28 16:16:59.290230
EPOCH #60       History Best Score: 88.82879699227331   Best Score: 88.8287969922
7331    Mean Score: 85.8848273454164
EPOCH 61 2020-06-28 16:21:04.381654
EPOCH #61       History Best Score: 88.82879699227331   Best Score: 88.7643955622
637     Mean Score: 87.15793181003373
EPOCH 62 2020-06-28 16:25:08.804796
EPOCH #62       History Best Score: 88.82879699227331   Best Score: 88.7643955622
637     Mean Score: 87.90899858774722
EPOCH 63 2020-06-28 16:29:14.181438
EPOCH #63       History Best Score: 88.82879699227331   Best Score: 87.7342746713
371     Mean Score: 85.14375350797508
EPOCH 64 2020-06-28 16:33:19.318772
EPOCH #64       History Best Score: 88.86558530151152   Best Score: 88.8655853015
1152    Mean Score: 86.24597237612683
EPOCH 65 2020-06-28 16:37:25.087996
EPOCH #65       History Best Score: 88.86558530151152   Best Score: 87.8160378706
1651    Mean Score: 86.03333186895695
EPOCH 66 2020-06-28 16:41:31.158693
EPOCH #66       History Best Score: 88.86558530151152   Best Score: 88.2347869470
6094    Mean Score: 86.38851359550586
EPOCH 67 2020-06-28 16:45:36.883918
EPOCH #67       History Best Score: 88.86558530151152   Best Score: 87.9797575054
1538    Mean Score: 85.6990920200806
EPOCH 68 2020-06-28 16:49:43.520287
EPOCH #68       History Best Score: 88.87357035135508   Best Score: 88.8735703513
5508    Mean Score: 86.08994915029817
EPOCH 69 2020-06-28 16:53:49.229907
EPOCH #69       History Best Score: 88.87357035135508   Best Score: 88.7193570793
7402    Mean Score: 85.63251330304021
EPOCH 70 2020-06-28 16:57:54.063383
EPOCH #70       History Best Score: 88.87357035135508   Best Score: 88.8735703513
5508    Mean Score: 87.0267125139359
EPOCH 71 2020-06-28 17:01:59.969467
EPOCH #71       History Best Score: 88.87357035135508   Best Score: 88.6441740502
4695    Mean Score: 86.89458622099653
EPOCH 72 2020-06-28 17:06:07.683909
EPOCH #72       History Best Score: 88.87357035135508   Best Score: 88.8700659860
2862    Mean Score: 86.5506811500659
EPOCH 73 2020-06-28 17:10:15.399061
EPOCH #73       History Best Score: 89.11860619222306   Best Score: 89.1186061922
2306    Mean Score: 86.82578436555995
EPOCH 74 2020-06-28 17:14:22.025022
EPOCH #74       History Best Score: 89.11860619222306   Best Score: 88.8735703513
5508    Mean Score: 87.45685764387574
EPOCH 75 2020-06-28 17:18:29.273152
EPOCH #75       History Best Score: 89.11860619222306   Best Score: 88.9258243184
```

```
5824    Mean Score: 87.45241632855445
EPOCH 76 2020-06-28 17:22:36.495790
EPOCH #76       History Best Score: 89.11860619222306   Best Score: 88.5302215028
0039    Mean Score: 86.89523287117409
EPOCH 77 2020-06-28 17:26:44.038318
EPOCH #77       History Best Score: 89.11860619222306   Best Score: 87.6143480826
7562    Mean Score: 86.11103824387888
EPOCH 78 2020-06-28 17:30:51.568751
EPOCH #78       History Best Score: 89.11860619222306   Best Score: 88.0177513138
6819    Mean Score: 86.51465759859272
EPOCH 79 2020-06-28 17:34:59.489474
EPOCH #79       History Best Score: 89.11860619222306   Best Score: 89.0575346312
2139    Mean Score: 85.79588752421981
EPOCH 80 2020-06-28 17:39:07.571052
EPOCH #80       History Best Score: 89.11860619222306   Best Score: 88.0652309019
4628    Mean Score: 86.30098478287653
EPOCH 81 2020-06-28 17:43:14.489581
EPOCH #81       History Best Score: 89.11860619222306   Best Score: 88.2217433472
5105    Mean Score: 86.2503433399394
EPOCH 82 2020-06-28 17:47:27.356423
EPOCH #82       History Best Score: 89.11860619222306   Best Score: 88.4211536234
2527    Mean Score: 86.75285376747692
EPOCH 83 2020-06-28 17:51:55.549779
EPOCH #83       History Best Score: 89.11860619222306   Best Score: 87.6124409116
4495    Mean Score: 85.79592660156439
EPOCH 84 2020-06-28 17:56:25.331640
EPOCH #84       History Best Score: 89.11860619222306   Best Score: 88.2956489895
9566    Mean Score: 86.36823472704074
EPOCH 85 2020-06-28 18:00:55.660039
EPOCH #85       History Best Score: 89.11860619222306   Best Score: 88.5747223649
9038    Mean Score: 87.11536535320298
EPOCH 86 2020-06-28 18:05:23.919105
EPOCH #86       History Best Score: 89.11860619222306   Best Score: 88.7986939196
3277    Mean Score: 86.96390032143574
EPOCH 87 2020-06-28 18:09:45.930127
EPOCH #87       History Best Score: 89.11860619222306   Best Score: 89.1186061922
2306    Mean Score: 86.8723812418915
EPOCH 88 2020-06-28 18:14:10.457606
EPOCH #88       History Best Score: 89.11860619222306   Best Score: 88.9967410969
2441    Mean Score: 87.43275615399054
EPOCH 89 2020-06-28 18:18:46.925493
EPOCH #89       History Best Score: 89.11860619222306   Best Score: 88.1068829888
516     Mean Score: 86.33229375410704
EPOCH 90 2020-06-28 18:23:07.954422
EPOCH #90       History Best Score: 89.11860619222306   Best Score: 89.1186061922
2306    Mean Score: 86.82676509324033
EPOCH 91 2020-06-28 18:27:28.554465
EPOCH #91       History Best Score: 89.11860619222306   Best Score: 88.5495580297
0309    Mean Score: 86.97312302021177
EPOCH 92 2020-06-28 18:31:48.423841
EPOCH #92       History Best Score: 89.11860619222306   Best Score: 89.1186061922
2306    Mean Score: 87.13046556723745
EPOCH 93 2020-06-28 18:36:13.218469
EPOCH #93       History Best Score: 89.11860619222306   Best Score: 89.0423310250
4605    Mean Score: 87.18467420031993
EPOCH 94 2020-06-28 18:40:38.253872
EPOCH #94       History Best Score: 89.11860619222306   Best Score: 88.8257580528
6686    Mean Score: 87.3912922650513
EPOCH 95 2020-06-28 18:46:08.798597
EPOCH #95       History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 87.39033473186399
```

```
EPOCH 96 2020-06-28 18:51:15.278981
EPOCH #96       History Best Score: 89.16698683259384   Best Score: 89.1186061922
2306    Mean Score: 87.75374116420379
EPOCH 97 2020-06-28 18:56:07.811912
EPOCH #97       History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 88.08210024137634
EPOCH 98 2020-06-28 19:01:09.151009
EPOCH #98       History Best Score: 89.16698683259384   Best Score: 88.8971983202
8861    Mean Score: 86.63314466942931
EPOCH 99 2020-06-28 19:06:09.225621
EPOCH #99       History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 87.11852306070229
EPOCH 100 2020-06-28 19:11:21.864490
EPOCH #100      History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 87.64049844504733
EPOCH 101 2020-06-28 19:16:34.751445
EPOCH #101      History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 88.3037057273194
EPOCH 102 2020-06-28 19:21:27.475411
EPOCH #102      History Best Score: 89.16698683259384   Best Score: 89.1669868325
9384    Mean Score: 87.93173720915874
EPOCH 103 2020-06-28 19:26:09.627751
EPOCH #103      History Best Score: 89.17310925712187   Best Score: 89.1731092571
2187    Mean Score: 88.55158691806017
EPOCH 104 2020-06-28 19:30:52.681106
EPOCH #104      History Best Score: 89.44781991734158   Best Score: 89.4478199173
4158    Mean Score: 88.75805959479375
EPOCH 105 2020-06-28 19:35:27.584793
EPOCH #105      History Best Score: 89.44781991734158   Best Score: 89.2180061153
5866    Mean Score: 88.82420121084472
EPOCH 106 2020-06-28 19:39:46.757104
EPOCH #106      History Best Score: 89.44781991734158   Best Score: 89.1669868325
9384    Mean Score: 88.34000483244208
EPOCH 107 2020-06-28 19:44:04.725604
EPOCH #107      History Best Score: 89.44781991734158   Best Score: 88.9891419051
3729    Mean Score: 87.96730809936336
EPOCH 108 2020-06-28 19:48:26.302106
EPOCH #108      History Best Score: 89.44781991734158   Best Score: 89.2529646709
4695    Mean Score: 87.86645572478491
EPOCH 109 2020-06-28 19:52:51.696032
EPOCH #109      History Best Score: 89.44781991734158   Best Score: 89.2710511529
1991    Mean Score: 88.33737703462555
EPOCH 110 2020-06-28 19:57:16.218659
EPOCH #110      History Best Score: 89.44781991734158   Best Score: 89.3836046576
9998    Mean Score: 88.55210363299491
EPOCH 111 2020-06-28 20:01:39.664912
EPOCH #111      History Best Score: 89.45135776875736   Best Score: 89.4513577687
5736    Mean Score: 88.76697286981621
EPOCH 112 2020-06-28 20:06:04.196439
EPOCH #112      History Best Score: 89.45135776875736   Best Score: 89.2195713596
2433    Mean Score: 88.38487665291483
EPOCH 113 2020-06-28 20:10:29.653079
EPOCH #113      History Best Score: 89.45135776875736   Best Score: 89.3885980787
5866    Mean Score: 88.70074735421494
EPOCH 114 2020-06-28 20:14:53.763725
EPOCH #114      History Best Score: 89.45135776875736   Best Score: 88.7590156687
2489    Mean Score: 88.04425212813314
EPOCH 115 2020-06-28 20:19:25.357362
EPOCH #115      History Best Score: 89.45135776875736   Best Score: 89.2124710798
4651    Mean Score: 88.36283922714105

EPOCH 116 2020-06-28 20:24:13.728862
```

```
EPOCH #116      History Best Score: 89.45135776875736   Best Score: 89.0484850923
9241    Mean Score: 88.41447955397759
EPOCH 117 2020-06-28 20:29:01.411540
EPOCH #117      History Best Score: 89.45135776875736   Best Score: 89.1745141368
131     Mean Score: 88.1386157127176
EPOCH 118 2020-06-28 20:34:06.849945
EPOCH #118      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.32229591380619
EPOCH 119 2020-06-28 20:38:53.708901
EPOCH #119      History Best Score: 89.53326873692622   Best Score: 89.0225462760
8575    Mean Score: 88.26293444744728
EPOCH 120 2020-06-28 20:43:45.597125
EPOCH #120      History Best Score: 89.53326873692622   Best Score: 89.0656775879
4476    Mean Score: 88.07829923230837
EPOCH 121 2020-06-28 20:48:48.839010
EPOCH #121      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.24399964609196
EPOCH 122 2020-06-28 20:53:50.719258
EPOCH #122      History Best Score: 89.53326873692622   Best Score: 89.1302375392
5783    Mean Score: 88.11850788443796
EPOCH 123 2020-06-28 20:59:24.084112
EPOCH #123      History Best Score: 89.53326873692622   Best Score: 88.8881638348
3923    Mean Score: 88.24645855420583
EPOCH 124 2020-06-28 21:04:43.567352
EPOCH #124      History Best Score: 89.53326873692622   Best Score: 89.2524270502
4726    Mean Score: 88.11007066486476
EPOCH 125 2020-06-28 21:10:05.259227
EPOCH #125      History Best Score: 89.53326873692622   Best Score: 89.2221488668
6453    Mean Score: 88.25402886808111
EPOCH 126 2020-06-28 21:15:18.474585
EPOCH #126      History Best Score: 89.53326873692622   Best Score: 89.2129558601
117     Mean Score: 88.15607893869374
EPOCH 127 2020-06-28 21:20:24.899806
EPOCH #127      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.33005295103075
EPOCH 128 2020-06-28 21:26:19.062571
EPOCH #128      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.30003454597012
EPOCH 129 2020-06-28 21:32:10.398857
EPOCH #129      History Best Score: 89.53326873692622   Best Score: 89.3621256355
0606    Mean Score: 88.37055400737862
EPOCH 130 2020-06-28 21:37:59.095151
EPOCH #130      History Best Score: 89.53326873692622   Best Score: 89.2589184361
812     Mean Score: 88.44037933902146
EPOCH 131 2020-06-28 21:43:45.278967
EPOCH #131      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.29619793105122
EPOCH 132 2020-06-28 21:49:29.825651
EPOCH #132      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.29224792424647
EPOCH 133 2020-06-28 21:55:22.645491
EPOCH #133      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.50618517254398
EPOCH 134 2020-06-28 21:59:56.157159
EPOCH #134      History Best Score: 89.53326873692622   Best Score: 89.1904740985
0994    Mean Score: 88.45031092909153
EPOCH 135 2020-06-28 22:04:26.688200
EPOCH #135      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.620524668649
EPOCH 136 2020-06-28 22:08:54.313146
EPOCH #136      History Best Score: 89.53326873692622   Best Score: 89.5332687369
```

```
2622    Mean Score: 88.71456174850228
EPOCH 137 2020-06-28 22:13:20.986082
EPOCH #137      History Best Score: 89.53326873692622   Best Score: 89.5332687369
2622    Mean Score: 88.91156909055776
EPOCH 138 2020-06-28 22:17:44.569072
EPOCH #138      History Best Score: 89.68895478241262   Best Score: 89.6889547824
1262    Mean Score: 89.24727874834528
EPOCH 139 2020-06-28 22:22:14.507237
EPOCH #139      History Best Score: 89.68895478241262   Best Score: 89.5867702912
6446    Mean Score: 89.11316880399056
EPOCH 140 2020-06-28 22:26:39.491162
EPOCH #140      History Best Score: 89.68895478241262   Best Score: 89.5772316712
0469    Mean Score: 89.10221846877637
EPOCH 141 2020-06-28 22:31:05.278515
EPOCH #141      History Best Score: 89.72509794681119   Best Score: 89.7250979468
1119    Mean Score: 89.30923736814084
EPOCH 142 2020-06-28 22:35:32.097352
EPOCH #142      History Best Score: 89.72509794681119   Best Score: 89.6889547824
1262    Mean Score: 89.28392411191972
EPOCH 143 2020-06-28 22:40:45.770600
EPOCH #143      History Best Score: 89.72509794681119   Best Score: 89.5752707381
0407    Mean Score: 89.19323989587977
EPOCH 144 2020-06-28 22:46:13.223989
EPOCH #144      History Best Score: 89.84716119373914   Best Score: 89.8471611937
3914    Mean Score: 89.25069317740899
EPOCH 145 2020-06-28 22:51:35.293303
EPOCH #145      History Best Score: 89.84716119373914   Best Score: 89.7824442571
8177    Mean Score: 89.24843192938552
EPOCH 146 2020-06-28 22:56:47.571284
EPOCH #146      History Best Score: 89.84716119373914   Best Score: 89.6839456935
5067    Mean Score: 89.23042728864361
EPOCH 147 2020-06-28 23:01:55.243854
EPOCH #147      History Best Score: 89.84716119373914   Best Score: 89.7250979468
1119    Mean Score: 89.3081821108709
EPOCH 148 2020-06-28 23:07:06.872162
EPOCH #148      History Best Score: 89.8833940639916    Best Score: 89.8833940639
916     Mean Score: 89.1029466836944
EPOCH 149 2020-06-28 23:11:35.462994
EPOCH #149      History Best Score: 89.8833940639916    Best Score: 89.8397871882
2223    Mean Score: 89.30150590630103
EPOCH 150 2020-06-28 23:16:12.918289
EPOCH #150      History Best Score: 89.8833940639916    Best Score: 89.6384039976
8608    Mean Score: 89.07833715760857
EPOCH 151 2020-06-28 23:21:01.502926
EPOCH #151      History Best Score: 89.97046035385273   Best Score: 89.9704603538
5273    Mean Score: 89.15221896031531
EPOCH 152 2020-06-28 23:26:20.048265
EPOCH #152      History Best Score: 89.97046035385273   Best Score: 89.5576395440
6078    Mean Score: 89.16621556193805
EPOCH 153 2020-06-28 23:31:53.320032
EPOCH #153      History Best Score: 89.97046035385273   Best Score: 89.6154110356
0185    Mean Score: 89.06760941272465
EPOCH 154 2020-06-28 23:36:44.405198
EPOCH #154      History Best Score: 89.97046035385273   Best Score: 89.8426301264
8218    Mean Score: 89.20183327122533
EPOCH 155 2020-06-28 23:42:03.463045
EPOCH #155      History Best Score: 89.97046035385273   Best Score: 89.5847718783
2077    Mean Score: 89.00166802198284
EPOCH 156 2020-06-28 23:46:54.786327
```

```
-
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-31ece2cd64e8> 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₩lib₩multiprocessing₩pool.py in map(self, func, iterable, chunk
size)
    266         in a list that is returned.
    267         '''
--> 268           return self._map_async(func, iterable, mapstar, chunksize).get()
    269
    270     def starmap(self, func, iterable, chunksize=None):

~₩anaconda3₩lib₩multiprocessing₩pool.py in get(self, timeout)
    649
    650     def get(self, timeout=None):
--> 651           self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

~₩anaconda3₩lib₩multiprocessing₩pool.py in wait(self, timeout)
    646
    647     def wait(self, timeout=None):
--> 648           self._event.wait(timeout)
    649
    650     def get(self, timeout=None):

~₩anaconda3₩lib₩threading.py in wait(self, timeout)
    550             signaled = self._flag
    551             if not signaled:
--> 552                     signaled = self._cond.wait(timeout)
    553             return signaled
    554

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

KeyboardInterrupt:
```

In [12]:

```
score_history = []
high_score_history = []
mean_score_history = []

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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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 156 2020-06-29 01:34:12.917482
EPOCH #156      History Best Score: 89.97046035385273   Best Score: 89.9704603538
5273    Mean Score: 89.51574597329774
EPOCH 157 2020-06-29 01:39:41.770176
EPOCH #157      History Best Score: 89.97046035385273   Best Score: 89.9704603538
5273    Mean Score: 89.47766181384623
EPOCH 158 2020-06-29 01:45:19.673681
EPOCH #158      History Best Score: 89.97046035385273   Best Score: 89.9704603538
5273    Mean Score: 89.2277282791253
EPOCH 159 2020-06-29 01:51:00.434454
EPOCH #159      History Best Score: 89.97046035385273   Best Score: 89.8026274598
8351    Mean Score: 89.20698023675587
EPOCH 160 2020-06-29 01:56:50.825406
EPOCH #160      History Best Score: 89.97046035385273   Best Score: 89.9704603538
5273    Mean Score: 89.26641924338028
EPOCH 161 2020-06-29 02:03:04.769320
EPOCH #161      History Best Score: 90.01004060418735   Best Score: 90.0100406041
8735    Mean Score: 89.16989351730815
EPOCH 162 2020-06-29 02:09:00.663398
EPOCH #162      History Best Score: 90.01004060418735   Best Score: 89.7288840992
```

In [18]:

```
score_history = []
high_score_history = []
mean_score_history = []

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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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 230 2020-06-29 09:01:03.729555
EPOCH #230      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.04602086951391
EPOCH 231 2020-06-29 09:07:47.658214
EPOCH #231      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.99917244813727
EPOCH 232 2020-06-29 09:13:17.500562
EPOCH #232      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.92797283510129
EPOCH 233 2020-06-29 09:18:48.413163
EPOCH #233      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.80381173076078
EPOCH 234 2020-06-29 09:24:12.343876
EPOCH #234      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.81414234291415
EPOCH 235 2020-06-29 09:29:42.754683
EPOCH #235      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.91747254985509
EPOCH 236 2020-06-29 09:35:19.741243
EPOCH #236      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.013696471027
EPOCH 237 2020-06-29 09:41:02.010081
EPOCH #237      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.99087322811737
EPOCH 238 2020-06-29 09:47:03.664275
EPOCH #238      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.93228378508323
EPOCH 239 2020-06-29 09:52:40.123932
EPOCH #239      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.75205501672308
EPOCH 240 2020-06-29 09:58:42.016191
EPOCH #240      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.7395917554176
EPOCH 241 2020-06-29 10:04:43.787605
EPOCH #241      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.88432041517834
EPOCH 242 2020-06-29 10:11:05.881937
EPOCH #242      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.88176304567351
EPOCH 243 2020-06-29 10:16:58.065143
EPOCH #243      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.15650530262972
EPOCH 244 2020-06-29 10:22:34.943983
EPOCH #244      History Best Score: 90.30612245524922   Best Score: 90.3061224552
```

```
4922    Mean Score: 89.84777331330721
EPOCH 245 2020-06-29 10:27:40.848569
EPOCH #245      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.98587739544584
EPOCH 246 2020-06-29 10:32:53.304694
EPOCH #246      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.01511460690863
EPOCH 247 2020-06-29 10:38:26.056673
EPOCH #247      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.91038346794844
EPOCH 248 2020-06-29 10:43:39.050764
EPOCH #248      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.96498404188158
EPOCH 249 2020-06-29 10:47:55.114593
EPOCH #249      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.88942349549919
EPOCH 250 2020-06-29 10:52:11.528696
EPOCH #250      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.81444827981343
EPOCH 251 2020-06-29 10:56:28.117851
EPOCH #251      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.96363914529982
EPOCH 252 2020-06-29 11:02:48.728190
EPOCH #252      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.06202362581844
EPOCH 253 2020-06-29 11:14:14.054438
EPOCH #253      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.0286999901385
EPOCH 254 2020-06-29 11:19:32.375707
EPOCH #254      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.97636488839149
EPOCH 255 2020-06-29 11:24:27.808951
EPOCH #255      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.0403245017101
EPOCH 256 2020-06-29 11:29:29.006380
EPOCH #256      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.03535066540545
EPOCH 257 2020-06-29 11:34:53.053442
EPOCH #257      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.88543645430855
EPOCH 258 2020-06-29 11:39:41.834175
EPOCH #258      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.0234415988778
EPOCH 259 2020-06-29 11:44:27.505525
EPOCH #259      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.92396221540281
EPOCH 260 2020-06-29 11:49:18.319662
EPOCH #260      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.88943690529547
EPOCH 261 2020-06-29 11:55:35.722310
EPOCH #261      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.77627882299775
EPOCH 262 2020-06-29 12:00:28.121561
EPOCH #262      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.69176389665026
EPOCH 263 2020-06-29 12:04:46.051533
EPOCH #263      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 90.10451394019755
EPOCH 264 2020-06-29 12:08:42.938918
EPOCH #264      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.99574003182099
```

```
EPOCH 265 2020-06-29 12:12:37.787374
EPOCH #265      History Best Score: 90.30612245524922   Best Score: 90.3061224552
4922    Mean Score: 89.97494762806357
EPOCH 266 2020-06-29 12:16:31.840568
EPOCH #266      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.05650282957455
EPOCH 267 2020-06-29 12:20:25.126320
EPOCH #267      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.96865061505346
EPOCH 268 2020-06-29 12:24:19.454519
EPOCH #268      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.08952690300063
EPOCH 269 2020-06-29 12:28:13.266130
EPOCH #269      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.84553378515999
EPOCH 270 2020-06-29 12:32:06.844085
EPOCH #270      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.957600850731
EPOCH 271 2020-06-29 12:36:01.689112
EPOCH #271      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.05316368276937
EPOCH 272 2020-06-29 12:39:55.076458
EPOCH #272      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 90.02710061463868
EPOCH 273 2020-06-29 12:43:50.041366
EPOCH #273      History Best Score: 90.33891639629465   Best Score: 90.2893216547
7808    Mean Score: 89.93065783576027
EPOCH 274 2020-06-29 12:47:44.119157
EPOCH #274      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.98831654983913
EPOCH 275 2020-06-29 12:51:38.643661
EPOCH #275      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.95728623509753
EPOCH 276 2020-06-29 12:55:34.596603
EPOCH #276      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.84663750509587
EPOCH 277 2020-06-29 12:59:29.159963
EPOCH #277      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.8386776466222
EPOCH 278 2020-06-29 13:03:32.377495
EPOCH #278      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.87560392469254
EPOCH 279 2020-06-29 13:07:29.288200
EPOCH #279      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.00591390449952
EPOCH 280 2020-06-29 13:11:58.659415
EPOCH #280      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 89.85394135879177
EPOCH 281 2020-06-29 13:17:51.469699
EPOCH #281      History Best Score: 90.33891639629465   Best Score: 90.3061224552
4922    Mean Score: 89.79866630128939
EPOCH 282 2020-06-29 13:23:29.045866
EPOCH #282      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.14133986752412
EPOCH 283 2020-06-29 13:28:50.820193
EPOCH #283      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 89.8301452221034
EPOCH 284 2020-06-29 13:33:48.953579
EPOCH #284      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.22530678941213
EPOCH 285 2020-06-29 13:38:55.398545
```

```
EPOCH #285      History Best Score: 90.33891639629465   Best Score: 90.3389163962
9465    Mean Score: 90.06579379081884
EPOCH 286 2020-06-29 13:44:39.018860

EPOCH #286      History Best Score: 90.3405550641873    Best Score: 90.3405550641873
Mean Score: 90.18724720690685
EPOCH 287 2020-06-29 13:49:47.159184
EPOCH #287      History Best Score: 90.3405550641873    Best Score: 90.3389163962946
5       Mean Score: 90.03349596615509
EPOCH 288 2020-06-29 13:55:27.714949
EPOCH #288      History Best Score: 90.3405550641873    Best Score: 90.3405550641873
Mean Score: 90.1891146554481
EPOCH 289 2020-06-29 13:59:49.021666
EPOCH #289      History Best Score: 90.3405550641873    Best Score: 90.3389163962946
5       Mean Score: 90.16735224570309
EPOCH 290 2020-06-29 14:04:10.290787
EPOCH #290      History Best Score: 90.3405550641873    Best Score: 90.3389163962946
5       Mean Score: 90.20372798481637
EPOCH 291 2020-06-29 14:08:30.152780
EPOCH #291      History Best Score: 90.3405550641873    Best Score: 90.3389163962946
5       Mean Score: 90.0987667134755
EPOCH 292 2020-06-29 14:12:50.483413
EPOCH #292      History Best Score: 90.3405550641873    Best Score: 90.3405550641873
Mean Score: 90.16230441742961
EPOCH 293 2020-06-29 14:17:11.483588
EPOCH #293      History Best Score: 90.3405550641873    Best Score: 90.3389163962946
5       Mean Score: 90.09807754518089
EPOCH 294 2020-06-29 14:21:31.218164
EPOCH #294      History Best Score: 90.3405550641873    Best Score: 90.3405550641873
Mean Score: 90.11435540967771
EPOCH 295 2020-06-29 14:25:51.183330
EPOCH #295      History Best Score: 90.3405550641873    Best Score: 90.3405550641873
Mean Score: 90.18553705082175
EPOCH 296 2020-06-29 14:30:10.411790
EPOCH #296      History Best Score: 90.3410699512117    Best Score: 90.3410699512117
Mean Score: 90.13065768215436
EPOCH 297 2020-06-29 14:34:29.834780
EPOCH #297      History Best Score: 90.3410699512117    Best Score: 90.3405550641873
Mean Score: 90.22482858348005
EPOCH 298 2020-06-29 14:38:49.154826
EPOCH #298      History Best Score: 90.3410699512117    Best Score: 90.3405550641873
Mean Score: 90.051683505459
EPOCH 299 2020-06-29 14:43:11.837915
EPOCH #299      History Best Score: 90.3410699512117    Best Score: 90.3410699512117
Mean Score: 90.06306747016897
EPOCH 300 2020-06-29 14:47:32.805022
EPOCH #300      History Best Score: 90.3410699512117    Best Score: 90.3389163962946
5       Mean Score: 89.96407862718095
```

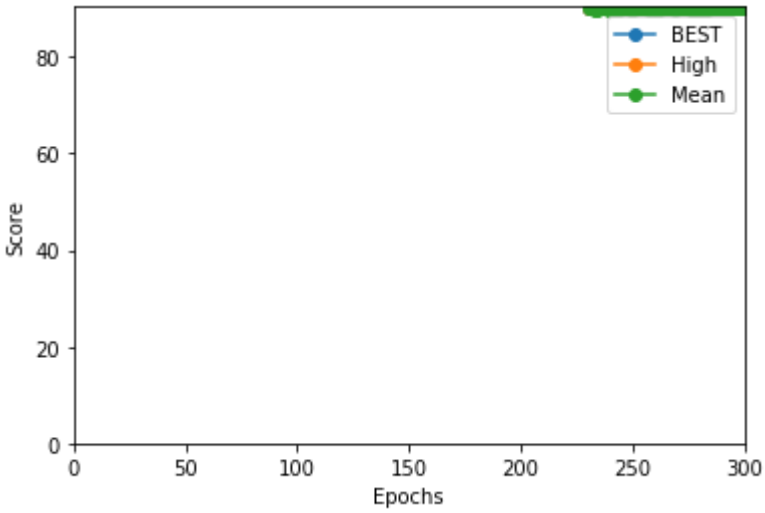
6. 결과 및 결언

Conclusion & Discussion

결과 그래프

In [19]:

```
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 [20]:

```
# 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_baseline_final3.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 [15]:

```
genomes
```

Out[15]:

```
array([[<module.genome.Genome object at 0x000002F79C03FD08>,
        <module.genome.Genome object at 0x000002F79A369108>,
        <module.genome.Genome object at 0x000002F7986D2F48>,
        <module.genome.Genome object at 0x000002F7A5D23548>,
        <module.genome.Genome object at 0x000002F7A5D238C8>,
        <module.genome.Genome object at 0x000002F7A5D23648>,
        <module.genome.Genome object at 0x000002F7A5D23188>,
        <module.genome.Genome object at 0x000002F7A5D23748>,
        <module.genome.Genome object at 0x000002F7A5D23F88>,
        <module.genome.Genome object at 0x000002F7986D2DC8>,
        <module.genome.Genome object at 0x000002F7A5D23C88>,
        <module.genome.Genome object at 0x000002F7A5D237C8>],
       [<module.genome.Genome object at 0x000002F792A0C788>,
        <module.genome.Genome object at 0x000002F79845DFC8>,
        <module.genome.Genome object at 0x000002F7A186A4C8>,
        <module.genome.Genome object at 0x000002F79238E388>,
        <module.genome.Genome object at 0x000002F796B43E08>,
        <module.genome.Genome object at 0x000002F79895B9C8>,
```

In [16]:

```
best_genomes
```

Out[16]:

```
[<module.genome.Genome at 0x2f79e8790c8>,
 <module.genome.Genome at 0x2f79e879d48>,
 <module.genome.Genome at 0x2f7a3a00cc8>,
 <module.genome.Genome at 0x2f79bd6be48>,
 <module.genome.Genome at 0x2f79e8793c8>,
 <module.genome.Genome at 0x2f79e879b48>,
 <module.genome.Genome at 0x2f79e879648>,
 <module.genome.Genome at 0x2f79e879108>,
 <module.genome.Genome at 0x2f7927ed0c8>,
 <module.genome.Genome at 0x2f7a41f1248>,
 <module.genome.Genome at 0x2f7a5d23908>,
 <module.genome.Genome at 0x2f7927ed2c8>,
 <module.genome.Genome at 0x2f7a7b6aec8>,
 <module.genome.Genome at 0x2f7969e2288>,
 <module.genome.Genome at 0x2f792378a88>,
 <module.genome.Genome at 0x2f79c626648>,
 <module.genome.Genome at 0x2f79238d248>,
 <module.genome.Genome at 0x2f79a465388>,
 <module.genome.Genome at 0x2f7986d2c08>,
 <module.genome.Genome at 0x2f7986d2588>,
 <module.genome.Genome at 0x2f7a7bf7c48>,
 <module.genome.Genome at 0x2f79845d9c8>,
 <module.genome.Genome at 0x2f79845d488>,
 <module.genome.Genome at 0x2f79845d888>,
 <module.genome.Genome at 0x2f79845da08>,
 <module.genome.Genome at 0x2f79845dc88>,
 <module.genome.Genome at 0x2f79c729688>,
 <module.genome.Genome at 0x2f79c729788>,
 <module.genome.Genome at 0x2f79c729fc8>,
 <module.genome.Genome at 0x2f79c729cc8>]
```

In [17]:

```
best_score_ever
```

Out[17]:

```
90.30612245524922
```

In [21]:

```
score_history = []
high_score_history = []
mean_score_history = []

while n_gen <= 500:
    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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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 301 2020-06-29 15:09:40.518149
EPOCH #301      History Best Score: 90.3410699512117    Best Score: 90.3410699512
117     Mean Score: 90.12878282980803
EPOCH 302 2020-06-29 15:14:17.834585
EPOCH #302      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.07296532838026
EPOCH 303 2020-06-29 15:18:36.444334
EPOCH #303      History Best Score: 90.34270861910434   Best Score: 90.3389163962
9465    Mean Score: 89.85864922645597
EPOCH 304 2020-06-29 15:22:55.496714
EPOCH #304      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 90.16695397628877
EPOCH 305 2020-06-29 15:27:17.834769
EPOCH #305      History Best Score: 90.34270861910434   Best Score: 90.3389163962
9465    Mean Score: 89.8798958275411
EPOCH 306 2020-06-29 15:31:36.696315
EPOCH #306      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.93219363377217
EPOCH 307 2020-06-29 15:35:54.764340
EPOCH #307      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 89.95037779374596
EPOCH 308 2020-06-29 15:40:13.882957
EPOCH #308      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 90.0226158442043
EPOCH 309 2020-06-29 15:44:34.009333
EPOCH #309      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 90.0529305857946
EPOCH 310 2020-06-29 15:48:52.914443
EPOCH #310      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 89.8151721476841
EPOCH 311 2020-06-29 15:53:11.103695
EPOCH #311      History Best Score: 90.34270861910434   Best Score: 90.3230320087
076     Mean Score: 89.7396032144188
EPOCH 312 2020-06-29 15:57:29.502737
EPOCH #312      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 89.7880209966996
EPOCH 313 2020-06-29 16:01:50.271537
EPOCH #313      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 89.86803482430403
EPOCH 314 2020-06-29 16:06:08.263267
EPOCH #314      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.8343327343727
EPOCH 315 2020-06-29 16:10:26.278851
EPOCH #315      History Best Score: 90.34270861910434   Best Score: 90.3389163962
```

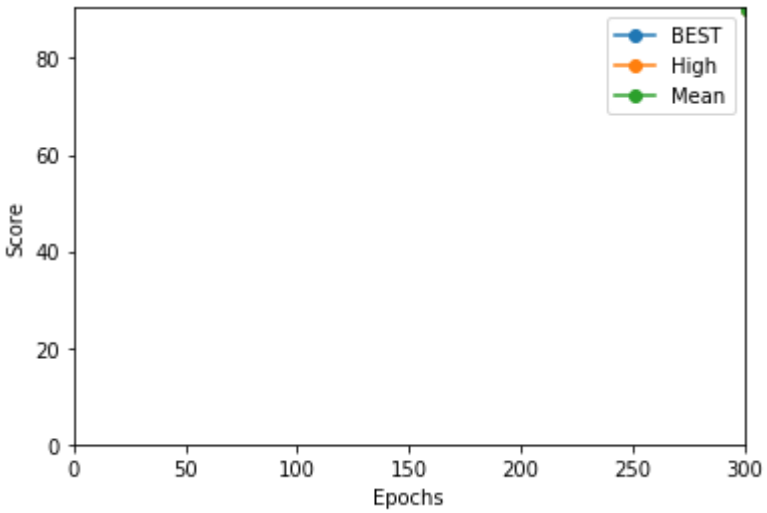
```
9465    Mean Score: 89.94693714415271
EPOCH 316 2020-06-29 16:14:43.618187
EPOCH #316      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.95359230471236
EPOCH 317 2020-06-29 16:19:06.342671
EPOCH #317      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.09225816976097
EPOCH 318 2020-06-29 16:23:25.337020
EPOCH #318      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.96188630213115
EPOCH 319 2020-06-29 16:27:46.443256
EPOCH #319      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.09953187061588
EPOCH 320 2020-06-29 16:32:05.546727
EPOCH #320      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.7582872080374
EPOCH 321 2020-06-29 16:36:24.225980
EPOCH #321      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.13956232452445
EPOCH 322 2020-06-29 16:40:41.762250
EPOCH #322      History Best Score: 90.34270861910434   Best Score: 90.3410699512
117     Mean Score: 89.99969576812634
EPOCH 323 2020-06-29 16:44:59.754129
EPOCH #323      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.98074671214219
EPOCH 324 2020-06-29 16:49:20.320543
EPOCH #324      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.11920772834364
EPOCH 325 2020-06-29 16:53:37.990140
EPOCH #325      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 89.96476400140949
EPOCH 326 2020-06-29 16:57:55.843466
EPOCH #326      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.13340426342431
EPOCH 327 2020-06-29 17:02:14.187025
EPOCH #327      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.24984714675477
EPOCH 328 2020-06-29 17:06:32.848363
EPOCH #328      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.0908758547785
EPOCH 329 2020-06-29 17:10:56.969039
EPOCH #329      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.13564892200904
EPOCH 330 2020-06-29 17:16:55.341540
EPOCH #330      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.09257992806944
EPOCH 331 2020-06-29 17:22:46.136999
EPOCH #331      History Best Score: 90.34270861910434   Best Score: 90.3427086191
0434    Mean Score: 90.08727471772609
EPOCH 332 2020-06-29 17:27:14.887857
EPOCH #332      History Best Score: 90.42033121736067   Best Score: 90.4203312173
6067    Mean Score: 90.01861821956615
EPOCH 333 2020-06-29 17:31:45.507304
EPOCH #333      History Best Score: 90.42033121736067   Best Score: 90.3427086191
0434    Mean Score: 90.09107379135955
EPOCH 334 2020-06-29 17:39:17.103655
EPOCH #334      History Best Score: 90.42033121736067   Best Score: 90.4004622923
5936    Mean Score: 90.24221903097401
EPOCH 335 2020-06-29 17:47:05.729591
EPOCH #335      History Best Score: 90.42033121736067   Best Score: 90.4203312173
6067    Mean Score: 90.15522320014574
```

```
EPOCH 336 2020-06-29 17:52:20.286602
EPOCH #336      History Best Score: 90.42033121736067   Best Score: 90.4203312173
6067    Mean Score: 90.24122882580873
EPOCH 337 2020-06-29 17:58:43.644759
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-9fdbd637672d> in <module>
     13         if __name__ == '__main__':
     14             pool = multiprocessing.Pool(processes=CPU_CORE)
---> 15                genomes[idx] = pool.map(genome_score, _genomes)
     16             pool.close()
     17             pool.join()

~₩anaconda3₩lib₩multiprocessing₩pool.py in map(self, func, iterable, chunk
size)
    266         in a list that is returned.
    267         '''
--> 268           return self._map_async(func, iterable, mapstar, chunksize).get()
    269
    270     def starmap(self, func, iterable, chunksize=None):

~₩anaconda3₩lib₩multiprocessing₩pool.py in get(self, timeout)
    649
    650     def get(self, timeout=None):
--> 651           self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

~₩anaconda3₩lib₩multiprocessing₩pool.py in wait(self, timeout)
    646
    647     def wait(self, timeout=None):
--> 648           self._event.wait(timeout)
    649
    650     def get(self, timeout=None):

~₩anaconda3₩lib₩threading.py in wait(self, timeout)
    550             signaled = self._flag
    551             if not signaled:
--> 552                 signaled = self._cond.wait(timeout)
    553             return signaled
    554

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

KeyboardInterrupt:
```

In [22]:

```
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()
```



In [23]:

```
# 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_baseline_final4.csv', index=False)
```

In [24]:

```
score_history = []
high_score_history = []
mean_score_history = []

while n_gen <= 500:
    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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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
```

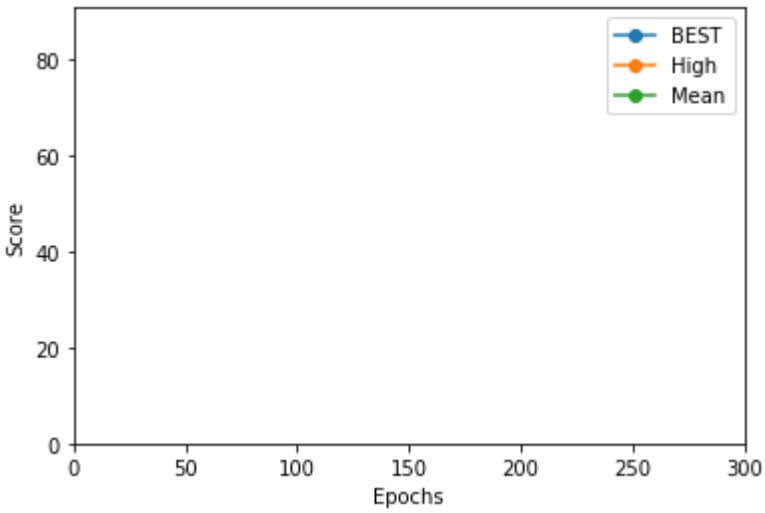
```
EPOCH 337 2020-06-29 21:04:18.481062
EPOCH #337      History Best Score: 90.42033121736067   Best Score: 90.4203312173606
7       Mean Score: 90.19500535009398
EPOCH 338 2020-06-29 21:08:24.393405
EPOCH #338      History Best Score: 90.42033121736067   Best Score: 90.3427086191043
4       Mean Score: 89.95632781099593
EPOCH 339 2020-06-29 21:12:28.171620
EPOCH #339      History Best Score: 90.42033121736067   Best Score: 90.4203312173606
7       Mean Score: 90.02561228085429
EPOCH 340 2020-06-29 21:16:29.761628
EPOCH #340      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 90.09746639333005
EPOCH 341 2020-06-29 21:20:26.843781
EPOCH #341      History Best Score: 90.68847676080763   Best Score: 90.4203312173606
7       Mean Score: 90.0732821405783
EPOCH 342 2020-06-29 21:24:12.434607
EPOCH #342      History Best Score: 90.68847676080763   Best Score: 90.4232355979862
9       Mean Score: 89.91526150024438
EPOCH 343 2020-06-29 21:27:58.498100
EPOCH #343      History Best Score: 90.68847676080763   Best Score: 90.3590940691835
Mean Score: 89.6247493842707
EPOCH 344 2020-06-29 21:31:42.585984
EPOCH #344      History Best Score: 90.68847676080763   Best Score: 90.5107925051373
5       Mean Score: 89.48309855693387
EPOCH 345 2020-06-29 21:35:28.998517
EPOCH #345      History Best Score: 90.68847676080763   Best Score: 90.3870663205823
Mean Score: 89.59764691100044
EPOCH 346 2020-06-29 21:39:14.141508
EPOCH #346      History Best Score: 90.68847676080763   Best Score: 90.5626194477990
2       Mean Score: 89.29436783946831
EPOCH 347 2020-06-29 21:43:11.100782
EPOCH #347      History Best Score: 90.68847676080763   Best Score: 90.2935089519990
7       Mean Score: 89.48178536461282
EPOCH 348 2020-06-29 21:47:03.503297
EPOCH #348      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.4610278140373
EPOCH 349 2020-06-29 21:50:57.679107
EPOCH #349      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.9161515850748
EPOCH 350 2020-06-29 21:54:47.315171
EPOCH #350      History Best Score: 90.68847676080763   Best Score: 90.3427086191043
4       Mean Score: 89.78214549367866
EPOCH 351 2020-06-29 21:58:36.049547
EPOCH #351      History Best Score: 90.68847676080763   Best Score: 90.5022750571951
6       Mean Score: 89.79108879865734
```

```
EPOCH 352 2020-06-29 22:02:37.111990
EPOCH #352      History Best Score: 90.68847676080763   Best Score: 90.6644733790198
2       Mean Score: 89.5937177946
EPOCH 353 2020-06-29 22:06:43.782721
EPOCH #353      History Best Score: 90.68847676080763   Best Score: 90.5637761003329
2       Mean Score: 89.30825453605944
EPOCH 354 2020-06-29 22:10:54.230878
EPOCH #354      History Best Score: 90.68847676080763   Best Score: 90.4812224450190
7       Mean Score: 89.68043108270973
EPOCH 355 2020-06-29 22:15:17.656637
EPOCH #355      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.67607762997133
EPOCH 356 2020-06-29 22:19:45.113756
EPOCH #356      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.59560625643032
EPOCH 357 2020-06-29 22:23:51.901102

---------------------------------------------------------------------------
-
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-24-9fdbd637672d> in <module>
     13         if __name__ == '__main__':
     14             pool = multiprocessing.Pool(processes=CPU_CORE)
---> 15                genomes[idx] = pool.map(genome_score, _genomes)
     16             pool.close()
     17             pool.join()

~₩anaconda3₩lib₩multiprocessing₩pool.py in map(self, func, iterable, chunk
size)
    266         in a list that is returned.
    267         '''
--> 268           return self._map_async(func, iterable, mapstar, chunksize).get()
    269
    270     def starmap(self, func, iterable, chunksize=None):

~₩anaconda3₩lib₩multiprocessing₩pool.py in get(self, timeout)
    649
    650     def get(self, timeout=None):
--> 651           self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

~₩anaconda3₩lib₩multiprocessing₩pool.py in wait(self, timeout)
    646
    647     def wait(self, timeout=None):
--> 648           self._event.wait(timeout)
    649
    650     def get(self, timeout=None):

~₩anaconda3₩lib₩threading.py in wait(self, timeout)
    550             signaled = self._flag
    551             if not signaled:
--> 552                 signaled = self._cond.wait(timeout)
    553             return signaled
    554

~₩anaconda3₩lib₩threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterr
upt)
    295             if timeout is None:
--> 296                 waiter.acquire()
```

```
297                 gotit = True
298             else:

KeyboardInterrupt:
```

In [25]:

```
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()
```



In [26]:

```
# 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_baseline_final5.csv', index=False)
```

In [27]:

```
score_history = []
high_score_history = []
mean_score_history = []

while n_gen <= 500:
    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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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
```

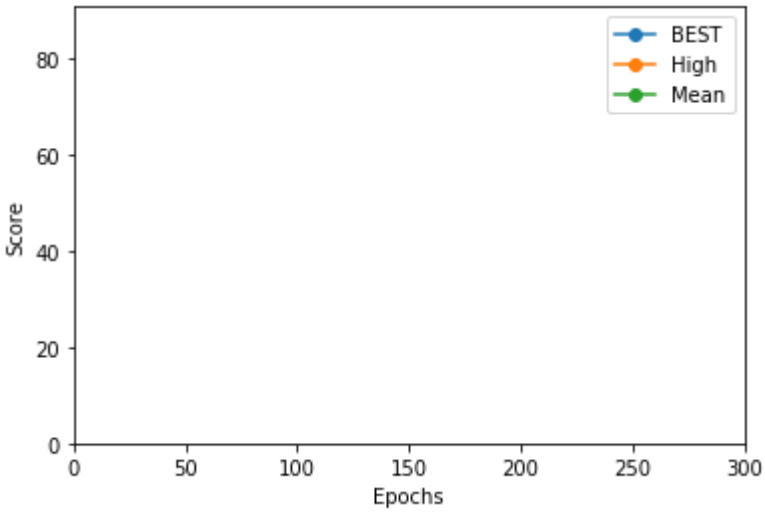
```
EPOCH 357 2020-06-29 22:25:54.701891
EPOCH #357      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.95713206084147
EPOCH 358 2020-06-29 22:30:21.133528
EPOCH #358      History Best Score: 90.68847676080763   Best Score: 90.5900417900250
3       Mean Score: 90.00417522379614
EPOCH 359 2020-06-29 22:34:44.012409
EPOCH #359      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.90069552757947
EPOCH 360 2020-06-29 22:39:06.486182
EPOCH #360      History Best Score: 90.68847676080763   Best Score: 90.4113836728587
3       Mean Score: 89.72120454378583
EPOCH 361 2020-06-29 22:43:36.367465
EPOCH #361      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.85381335898349
EPOCH 362 2020-06-29 22:47:42.689133
EPOCH #362      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 89.87860500024117
EPOCH 363 2020-06-29 22:51:59.517156
EPOCH #363      History Best Score: 90.68847676080763   Best Score: 90.6884767608076
3       Mean Score: 90.19770429816995
EPOCH 364 2020-06-29 22:56:12.278139
```

```
---------------------------------------------------------------------------
----
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-27-9fdbd637672d> in <module>
     13         if __name__ == '__main__':
     14             pool = multiprocessing.Pool(processes=CPU_CORE)
---> 15                genomes[idx] = pool.map(genome_score, _genomes)
     16             pool.close()
     17             pool.join()

~₩anaconda3₩lib₩multiprocessing₩pool.py in map(self, func, iterable, chu
nksize)
    266         in a list that is returned.
    267         '''
--> 268           return self._map_async(func, iterable, mapstar, chunksize).ge
t()
    269
    270     def starmap(self, func, iterable, chunksize=None):

~₩anaconda3₩lib₩multiprocessing₩pool.py in get(self, timeout)
    649
    650     def get(self, timeout=None):
```

```
--> 651             self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

~₩anaconda3₩lib₩multiprocessing₩pool.py in wait(self, timeout)
    646
    647     def wait(self, timeout=None):
--> 648         self._event.wait(timeout)
    649
    650     def get(self, timeout=None):

~₩anaconda3₩lib₩threading.py in wait(self, timeout)
    550             signaled = self._flag
    551             if not signaled:
--> 552                 signaled = self._cond.wait(timeout)
    553             return signaled
    554

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

KeyboardInterrupt:
```

In [28]:

```
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()
```



In [29]:

```
# 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_baseline_final10.csv', index=False)
```

In [2]:

```
score_history = []
high_score_history = []
mean_score_history = []
n_gen = 363

while n_gen <= 500:
    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 #%sWtHistory Best Score: %sWtBest Score: %sWtMean 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
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call last)
<ipython-input-2-dd0d3fc2d331> in <module>
      5
      6 while n_gen <= 500:
----> 7     print('EPOCH', n_gen, datetime.datetime.now())
      8     genomes = np.array(genomes)
      9     while len(genomes)%CPU_CORE != 0:

NameError: name 'datetime' is not defined
```

In [ ]: