بررسی کارکرد Evolution Strategies

درس رایانش تکاملی – مدرس: دکتر روحانی

نویسنده: آریا عربان

مقدمه

Evolution Strategies (به مخفف: ES) یک الگوریتم بهینه سازی است که عضوی از خانواده الگوریتم‌های تکاملی می‌باشد. این الگوریتم در دهه 1960 توسط سه دانشجو آلمانی پیشنهاد شد.

ایده اصلی این الگوریتم آن است که با ایجاد تغییرات تدریجی در یک جمعیت اولیه، می‌توان به جواب بهینه رسید.

نسخه پیشنهادی اولیه این الگوریتم به‌این گونه بود که جمعیت اولیه تنها شامل یک عضو است، و دائم بر روی یک عضو جهش اتفاق می‌افتد. در هر نسل، در صورتی که فرزند جهش یافته نتیجه بهتری از والد خود بگیرد، آن فرزند به عنوان والد نسل بعد انتخاب می‌شود (در این حالت به اصطلاح جهش موفق رخ داده است). در غیر این صورت، خود والد همچنان برای نسل بعد انتخاب می‌شود.

قابل بیان است که این الگوریتم مورد ویرایش‌هایی قرار گرفته، و امروزه این حالت‌های ویرایش یافته بیشتر مورد توجه هستند.

اولین روش ویرایش‌یافته، آن است که در هر نسل، از یک والد چند فرزند تولید می‌شود و این فرزندان، والدشان را در نسل بعدی جایگزین می‌کنند. این روش را با $ES(\mu, \lambda)$ نشان می‌دهند، که در آن μ جمعیت والدین و μ جمعیت فرزندان است.

روش ویرایش‌یافته دوم، همانند روش قبل می‌باشد، با این تفاوت که در هر نسل، ممکن است که هم والدین و هم فرزندان به نسل بعد منتقل شوند. این روش را با $ES(\mu+\lambda)$ نشان می‌دهند.

هدف و راه حل

در تمرین این هفته، خواسته شده که الگوریتم کلاسیک ES پیاده سازی شود، تا از آن جهت پیدا کردن مقادیر بهینه تابع تست Rosenbrock استفاده شود.

برای انجام این کار، ابتدا در زبان پایتون یک تابعی جهت انجام ES پیاده سازی کرده‌ام که پارامترهای تابع هدف، بازه عددی، تعداد جهش، میزان جهش (σ) و تعداد متغیر های تصمیم (N) را به عنوان ورودی می‌گیرد، و در نهایت نتیجه بهترین امتیاز پس از اعمال ES، به همراه نرخ موفقیت را در خروجی می‌دهد.

در فایل اجرایی ES نیز یک تابع دیگری جهت پیاده سازی تابع تست Rosenbrock نیز نوشته‌ام، که این تابع به عنوان تابع هدف ورودی ES قرار می‌گیرد.

در ادامه به بررسی نتیجه تغییر مقادیر N و σ بر عملکرد الگوریتم ES پرداخته‌ام.

شرح نتایج

تابع تست Rosenbrock به شکل زیر می‌باشد:

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \quad \text{where} \quad \mathbf{x} = (x_1, \ldots, x_N) \in \mathbb{R}^N$$

با بررسی تابع فوق، می‌توان به راحتی متوجه شد مقدار بهینه این تابع به ازای ورودی‌ای خواهد بود که تمامی عناصر آن برابر مقدار عددی 1 باشد. با این دانش، به بررسی نتایج حاصل از اعمال الگوریتم ES به ازاء مقادیر مختلف N (تعداد متغیر های تصمیم برای ورودی) و σ (میزان جهش) پرداخته می‌شود.

با بررسی N به ازاء مقادیر مختلف 2, 5, 10, 50 و در نظر گرفتن سیگما دارای مقدار ثابت σ=0.15، امتیازهای نهایی پس از 10000 جهش، همانند شکل زیر بدست آمده‌اند:

```
N: 2 --- best_score: 0.0001
N: 5 --- best_score: 0.71954
N: 10 --- best_score: 9.63751
N: 50 --- best_score: 291.46032
```

متوجه می‌شویم که هرچه N را بزرگتر فرض می‌کنیم، گرفتن امتیاز بهتر برای الگوریتم، دشوارتر می‌شود، علت آن هم این است که تعداد پارامتر هایی که الگوریتم به دنبال تنظیم کردن آن است، افزایش پیدا می‌کند. به همین جهت نیز احتمال آنکه الگوریتم در local minima بیفتد نیز بسیار افزایش می‌یابد. همچنین دقت شود که معمولا در لحظه ای که الگوریتم را شروع می‌کنیم، هرچه N بزرگتر باشد، امتیازی که الگوریتم به دنبال کاهش آن است هم خیلی بزرگتر می‌شود. مثلا در حالتی که N=50 بوده، امتیاز اولیه 12 برابر بیشتر از حالتی که N=2 بوده، بدست آمده.

برای ادامه کار، با در نظر گرفتن N=20، به بررسی σ به ازاء مقادیر مختلف پرداخته می‌شود.

```
σ: 0.02 --- best_score: 75020.75 --- success_rate: 0.4963
σ: 0.04 --- best_score: 19.73 --- success_rate: 0.2874
σ: 0.06 --- best_score: 18.41 --- success_rate: 0.1927
σ: 0.15 --- best_score: 33.21 --- success_rate: 0.0739
σ: 0.5 --- best_score: 378.84 --- success_rate: 0.0256
σ: 1 --- best_score: 1391.46 --- success_rate: 0.0134
σ: 5 --- best_score: 311313.38 --- success_rate: 0.0027
```

با بررسی σ به ازاء مقادیر بالا، مشاهده می‌شود که اگر σ را خیلی کوچک فرض کنیم، بهترین امتیاز عدد پرتی می‌شود. علت آن، این است که الگوریتم در local minima قرار می‌گیرد، و زمانی که σ مقداری خیلی پایین داشته باشد، الگوریتم از این حالت نمی‌تواند خارج شود. همچنین مشاهده می‌شود که اگر σ را خیلی بزرگ فرض کنیم، بهترین امتیاز همچنان عدد پرتی خواهد شد، چراکه در این حالت امکان بررسی نشدن مینمم‌ها و اطراف آن به دلیل بزرگ بودن گام‌ها، بسیار زیاد می‌شود.

در نهایت از مقادیر بالا، صحت قانون نسبت 1 به 5 نیز قابل ملاحظه است، چراکه هرچه امتیاز بهتر باشد، متوجه می‌شویم که نرخ موفقیت نیز به 0.2 نزدیکتر می‌باشد. به عنوان مثال، دیده می‌شود که مقدار $\sigma = 0.06$ بهترین نتیجه را می‌دهد.

اگر به نرخ موفقیت آن نیز دقت کنیم، متوجه می‌شویم که مقدار آن نیز بسیار نزدیک به $0.2 = 1/5$ می‌باشد

ضمیمه (کد) :

```python
def es_classic(objective, bounds, n_iter, step_size, n_parent_size):
    best_score = 1e+10
    success_rate = 0  # The ratio between successful mutations and total number of iterations

    best_values = None
    # initial values
    parent_values = ([random.uniform(bounds[0], bounds[1]) for _ in range(n_parent_size)])

    # perform the search
    for epoch in range(n_iter):
        # evaluate fitness for the population
        parent_score = objective(parent_values)

        # check if this parent is the best solution ever seen
        if parent_score < best_score:
            best_score = parent_score  # we just use this info and print it
            best_values = parent_values
            if len(best_values) <= 4:
                print(f'epoch {epoch} -- Best Values: {best_values} -- Best score:({best_score:.5f})')
            else:
                print(
                    f"epoch {epoch} -- Best Values: [{str(best_values[0:2]).strip('[]')}, ..., {str(best_values[2:4]).strip('[')} -- Best score:({best_score:.5f})")
        # create children for parent
        child_values = None
        while child_values is None or not _in_bounds(child_values, bounds):
            child_values = parent_values + randn(n_parent_size) * step_size

        if objective(child_values) < parent_score:
            parent_values = child_values
            success_rate = success_rate + 1
    success_rate = success_rate / n_iter

    return best_score, success_rate
```

```python
def _in_bounds(point, bounds):
    # enumerate all dimensions of the point
    for xi in point:
        if xi < bounds[0] or xi > bounds[1]:
            return False
    return True
```

شکل 1 – پیاده سازی الگوریتم ES کلاسیک

```python
def rosenbrock(v):
    """
      This function returns the Rosenbrock function result of a vector
    """
    total = 0
    for i in range(len(v) - 1):
        xi = v[i]
        xnext = v[i + 1]
        new = 100 * (xnext - xi ** 2) ** 2 + (xi - 1) ** 2
        total = total + new
    return total


def main():
    random.seed(103)
    # define the maximum step size
    sigma = 0.06

    # define the parent vector size
    n = 20

    best_score, success_rate = es_classic(rosenbrock, bounds=[-30, 30], n_iter=10000, step_size=sigma, n_parent_size=n)
    print('\n ---- \n')
    print(f"best_score: {best_score:.2f} --- success_rate: {success_rate}")


if __name__ == '__main__':
    main()
```

شکل 2 – فایل اجرایی ES

```
epoch 1 -- Best Values: [28.73209744 -2.58311214, ..., 13.071809   14.10635271] -- Best score:(439135309.58697)
epoch 3 -- Best Values: [28.76589483 -2.64359119, ..., 13.14287092 14.22001854] -- Best score:(438161064.17817)
epoch 4 -- Best Values: [28.75983362 -2.65088117, ..., 13.10934827 14.15862795] -- Best score:(437737872.18032)
epoch 5 -- Best Values: [28.69302169 -2.62842439, ..., 12.97092765 14.07973925] -- Best score:(435960069.58579)
epoch 6 -- Best Values: [28.61132531 -2.49013659, ..., 13.04959902 14.11030873] -- Best score:(435436837.41514)
epoch 7 -- Best Values: [28.63411184 -2.52227309, ..., 13.10718698 13.94729682] -- Best score:(435256971.53281)
epoch 8 -- Best Values: [28.66986028 -2.51443091, ..., 13.06691362 13.92068899] -- Best score:(433742352.09647)
epoch 9 -- Best Values: [28.77200039 -2.44444708, ..., 12.9880378  13.92162226] -- Best score:(432870722.08802)
epoch 10 -- Best Values: [28.79976463 -2.39835645, ..., 12.98531629 13.89904584] -- Best score:(431489881.53040)
epoch 11 -- Best Values: [28.77780575 -2.35754629, ..., 13.16317862 13.90550183] -- Best score:(430101649.61932)
epoch 17 -- Best Values: [28.68707715 -2.379827  , ..., 13.23217758 13.98792684] -- Best score:(428617476.23352)
epoch 18 -- Best Values: [28.74542794 -2.32407201, ..., 13.23149095 13.96726746] -- Best score:(427499982.70079)
epoch 19 -- Best Values: [28.79560358 -2.20548587, ..., 13.27505401 13.94257381] -- Best score:(425665220.68714)
epoch 21 -- Best Values: [28.71829226 -2.22084072, ..., 13.32963948 13.90961364] -- Best score:(424839473.33396)
epoch 23 -- Best Values: [28.58921059 -2.30301127, ..., 13.3515455  14.00714975] -- Best score:(421241578.79770)
epoch 28 -- Best Values: [28.50947753 -2.29789884, ..., 13.3707126  14.07377432] -- Best score:(420659452.72825)
```

....

```
epoch 4100 -- Best Values: [0.87766487 0.79011127, ..., 0.67970258 0.43559694] -- Best score:(32.90721)
epoch 4130 -- Best Values: [0.87010366 0.76982597, ..., 0.66139001 0.45067533] -- Best score:(28.50166)
epoch 4137 -- Best Values: [0.85010064 0.70045523, ..., 0.62771098 0.414794  ] -- Best score:(25.81383)
epoch 4262 -- Best Values: [0.86771866 0.66175127, ..., 0.5383764  0.35238586] -- Best score:(25.71881)
epoch 4292 -- Best Values: [0.82936049 0.69509429, ..., 0.56048036 0.30138228] -- Best score:(25.54035)
epoch 4323 -- Best Values: [0.8655617  0.78950654, ..., 0.63791165 0.27623336] -- Best score:(25.19351)
epoch 4372 -- Best Values: [0.93473978 0.82709244, ..., 0.58661263 0.2155283 ] -- Best score:(24.78908)
epoch 4413 -- Best Values: [0.96055501 0.79752802, ..., 0.5089391 0.2607823] -- Best score:(24.44748)
epoch 4437 -- Best Values: [0.9279314  0.78286234, ..., 0.56577521 0.25962719] -- Best score:(21.81548)
epoch 4616 -- Best Values: [0.88442958 0.79445036, ..., 0.5697123  0.30919307] -- Best score:(21.06913)
epoch 4929 -- Best Values: [0.93578052 0.82115247, ..., 0.63972671 0.41527781] -- Best score:(20.39119)
epoch 4975 -- Best Values: [0.8745613  0.81330081, ..., 0.69938796 0.48364072] -- Best score:(19.72444)
epoch 5369 -- Best Values: [0.87258736 0.7635098 , ..., 0.64698101 0.39009204] -- Best score:(18.67446)
epoch 5750 -- Best Values: [0.84902417 0.73078343, ..., 0.59122045 0.40991231] -- Best score:(18.53596)
epoch 7440 -- Best Values: [0.83457442 0.74335548, ..., 0.60730304 0.43942907] -- Best score:(18.41383)

 ----

best_score: 18.41 --- success_rate: 0.1927
```

شکل3 – نتیجه نمونه از اجرا کد