دانشکده مهندسی

گروه مهندسی کامپیوتر

گرایش هوش مصنوعی

الگویتم استراتژی تکاملی

استاد:

دکتر مجتبی روحانی

دانشجو:

علی گلی

آبان ماه

سال ۱۴۰۲

۱ الگویتم ES

الگوریتم استراتژی های تکاملی با توابع بنچمارک؛ با یک جمعیت یک نفره انجام شده است. نکته قابل توجه میزان step یا sigma می باشد که در کد به صورت یک عدد کوچکتر از یک بیان شده است.

ابتدا یک جمعیت تک نفره به صورت تصادفی از روی یک رشته از اعداد (بازه اعداد قابل تعیین است) ساخته می‌شود. سپس جهش روی همین فرد انجام می‌شود. در صورتی که جهش باعث بهتر شدن ژن مربوطه بشود جهش نگاه داشته می شود. در غیر اینصورت همان نسل قبلی باقی می‌ماند.

عملیات جهش به صورت اتفاقی یکی از اعداد بازه اعداد ژن تعریف شده را انتخاب می‌کند.

عملیات evaluation که برتری ژن را پیدا می‌کند؛ ابتدا باید یک تابع بنچمارک به عنوان ورودی بگیرد و سپس مقدار برتری ژن را محاسبه می‌کند. در صورت خارج شدن عدد از مقدار مشخص شده تابع بنچمارک؛ نرمال می‌شود.

۲ کد الگوریتم ES

۱.۲ کتابخانه های مورد نیاز

ابتدا کتابخانه های مورد نیاز را به پروژه اضافه می‌کنیم:

```
1  import numpy as np
2  from typing import List,
   Callable, Tuple
3  import random
4  from matplotlib import pyplot as
    plt
5
```

۲.۲ توابع بنچمارک

حال توابع بنچمارک که احتیاج داریم تعریف می‌کنیم. این توابع ژن را دریافت و یک اسکالر باز می‌گردانند.

```
1  # BENCHMARK FUNCTIONS DEFINITION
2  def benchmarkFunction1(individual: List)-> int:
3      return (np.sum(individual)) ** 2
4  def benchmarkFunction2(individual: List)-> int:
5      sum = 0
6      for index in range(len(individual)):
7          if index + 1 == len(individual):break
8          sum+= 100 * (individual[index+1] - individual[index]**2)**2 +
    (individual[index] - 1)**2
9      return sum
```

۳.۲ تابع ژن

حال تابع ژن را باید تعریف کنیم و اعداد مجازی که درون ژن می‌توانند قرار بگیرند را مشخص کنیم.

```python
# GENE DEFINITION
def genes():
    return (0,1)
```

۴.۲ ساخت جمعیت تک نفره

یک جمعیت تک نفره که به صورت اتفاقی ژنی با سایز geneSize دارد می‌سازیم.

```python
# CREATE RANDOM POPULATION
def population_create(populationSize: int, geneSize: int) -> List:
    population = []
    for index in range(populationSize):
        individual = [random.choice(genes()) for _ in range(geneSize)]
        population.append(individual)
    return population
```

۵.۲ تابع جهش

تابع جهشی باید نوشته شود که بر اساس سیگما جهش دهد. این تابع ژن یک فرد را دریافت و نوکلئوتید های ژن را؛ اگر سیگما مجاز بداند؛ تغییر خواهد داد.

```python
# MUTATION
def mutate_individual(individual: List, mutation_rate) -> List:
    mutated_individual = []
    for gene in individual:
        if random.random() < mutation_rate:
            mutated_individual.append(random.choice(genes()))
        else:
            mutated_individual.append(gene)
    return mutated_individual
```

۶.۲ تابع نرمال

این تابع اگر میزان امتیاز ژن بیش از حد شود آن را به مقدار ماکسیمم؛ و اگر کمتر از حد شود آن را به مقدار مینیمم، تخمین می‌زند.

```python
# NORAMLIZATION THE EVALUATION
def normalization(bound: Tuple, individual: List, func: Callable)-> int:
    score = func(individual)
    if score < bound[0]:    return bound[0]
    elif score > bound[1]:  return bound[1]
    else:                   return score
```

۷.۲ تابع ارزیابی

این تابع هر فردی از جامعه را دریافت و برای آن یک امتیاز برمی‌گرداند.

```python
# Evaluation
def evaluation(individual: List, func: Callable, bound: Tuple)->int:
    return normalization(bound=bound, func=func, individual=individual)
```

۸.۲ توابع ES

این توابع الگوریتم اصلی ES را اجرا می‌کنند. تابع اول این امکان را به شما می‌دهد که بتوانید به میزان geneSize های متفاوت کد خود را ارزیابی کنید. همچنین در نهایت نتایج را به شما نشان می‌دهد.

```python
# EVOLUTIONAR STRATEGY
def ES(func: callable, bound:Tuple, sigma:int=.5):
    # CHEKC IT FOR DIFFRENT SIZE OF GENES
    N = [2, 5, 10, 50]
    generations = 5
    for n in N:
        x,y=ES_SUB(n, generations, func, bound, sigma)
        plt.plot(x,y, label=f'N={n}')
    plt.title(f"ES for sigma={sigma}")
    plt.legend()
    plt.show()
def ES_SUB(N:int, generations: int, func: callable, bound:Tuple, sigma):
    x,y = [0], [0]
    population = population_create(populationSize=1, geneSize=N)
    individual = population[0]
    for generation in range(generations):
        new_individual = mutate_individual(individual, mutation_rate=sigma)
        new_score = evaluation(individual=new_individual, func=func, bound=bound)
        score = evaluation(individual=individual, func=func, bound=bound)
        if new_score > score:
            individual = new_individual
            y.append(new_score)
        else:
            y.append(score)
        x.append(x[-1]+1)
    return x,y
```

۹.۲ مقدار سیگما

قبل از اجرای الگوریتم اصلی می‌توانید یک حلقه بنویسید تا مقادیر مختلف سیگما را نیز به شما نشان دهد.

```python
# CHECK FOR DIFFRENT SIGMAS
for _ in range(10,100,10):
    ES(benchmarkFunction2, (-30, 30), sigma=_/100)
```

۱۰.۲ کد نهایی

در نهایت کد نهایی به شکل زیر خواهد بود.

```python
import numpy as np
from typing import List, Callable, Tuple
import random
from matplotlib import pyplot as plt

# BENCHMARK FUNCTIONS DEFINITION
def benchmarkFunction1(individual: List)-> int:
    return (np.sum(individual)) ** 2
def benchmarkFunction2(individual: List)-> int:
    sum = 0
    for index in range(len(individual)):
        if index + 1 == len(individual):break
        sum+= 100 * (individual[index+1] - individual[index]**2)**2 + (individual[index] - 1)**2
    return sum

# GENE DEFINITION
def genes():
    return (0,1)


# CREATE RANDOM POPULATION
def population_create(populationSize: int, geneSize: int) -> List:
    population = []
    for index in range(populationSize):
        individual = [random.choice(genes()) for _ in range(geneSize)]
        population.append(individual)
    return population

# MUTATION
def mutate_individual(individual: List, mutation_rate) -> List:
    mutated_individual = []
    for gene in individual:
        if random.random() < mutation_rate:
            mutated_individual.append(random.choice(genes()))
        else:
            mutated_individual.append(gene)
    return mutated_individual

# NORAMLIZATION THE EVALUATION
def normalization(bound: Tuple, individual: List, func: Callable)-> int:
    score = func(individual)
    if score < bound[0]:    return bound[0]
    elif score > bound[1]:  return bound[1]
    else:                   return score

# Evaluation
def evaluation(individual: List, func: Callable, bound: Tuple)->int:
    return normalization(bound=bound, func=func, individual=individual)

# EVOLUTIONAR STRATEGY
def ES(func: callable, bound:Tuple, sigma:int=.5):
    # CHEKC IT FOR DIFFRENT SIZE OF GENES
    N = [2, 5, 10, 50]
    generations = 5
    for n in N:
        x,y=ES_SUB(n, generations, func, bound, sigma)
        plt.plot(x,y, label=f'N={n}')
    plt.title(f"ES for sigma={sigma}")
    plt.legend()
    plt.show()
def ES_SUB(N:int, generations: int, func: callable, bound:Tuple, sigma):
    x,y = [0], [0]
    population = population_create(populationSize=1, geneSize=N)
    individual = population[0]
    for generation in range(generations):
        new_individual = mutate_individual(individual, mutation_rate=sigma)
        new_score = evaluation(individual=new_individual, func=func, bound=bound)
        score = evaluation(individual=individual, func=func, bound=bound)
        if new_score > score:
            individual = new_individual
            y.append(new_score)
        else:
            y.append(score)
        x.append(x[-1]+1)
    return x,y

# CHECK FOR DIFFRENT SIGMAS
for _ in range(10,100,10):
    ES(benchmarkFunction2, (-30, 30), sigma=_/100)
```

۳ نتایج

در نهایت می‌توانیم نتایج زیر را مشاهده کنیم: