Go语言如何实现遗传算法

作者：Diwei翻译 2017-11-16 15:25:54

开发 # 后端 # 大数据 # 算法

本文将重点介绍如何用Go语言实现遗传算法。如果你还没有参加过GoLang Tour，我还建议你快速看一下这门语言的介绍。话不多说，让我们开始从代码说起吧!

出于好玩的心态，我决定学习一下Go语言。我认为学习新语言***的方法就是深入学习，并且尽可能多犯错误。这样做虽然可能会很慢，但是可以确保在后面的过程中再也不会出现编译的错误。

Go语言与我习惯的其他语言不同。Go更喜欢自己单独实现，而其他像Java这类语言更喜欢继承。其实在Go语言里面根本没有继承这种概念，因为它压根就没有对象这一说法。比如说C语言，它有结构体，但是没有类。但是这样它还是可以有像“构造者”这样的常见思想和设计模式(一种在这种情况下有序地产生结构体的方式)。

Go语言坚决拥护组合(composition)，同时也很反对继承的做法，在网络上引起了强烈的讨论，同时也让人们重新思考了语言该往哪个方向发展。所以，从这个角度来看，Go语言与其它语言的差别可能也没有那么大。

本文将重点介绍如何用Go语言实现遗传算法。如果你还没有参加过GoLang Tour，我还建议你快速看一下这门语言的介绍。

话不多说，让我们开始从代码说起吧!***个例子与我以前做过的很类似：找到一个二次的最小值。

复制

```
type GeneticAlgorithmSettings struct {
  PopulationSize int
  MutationRate int
  CrossoverRate int
  NumGenerations int
  KeepBestAcrossPopulation bool
}

type GeneticAlgorithmRunner interface {
  GenerateInitialPopulation(populationSize int) []interface{}
  PerformCrossover(individual1, individual2 interface{}, mutationRate int) interface{}
  PerformMutation(individual interface{}) interface{}
  Sort([]interface{})
}
```

我立马定义了一组设置，以便在稍后启动的算法中用到。

相似话题的热门内容

Java前端不语句的简要指南
19597内容

开发工具
6795内容

测试
390内容

游戏开发

51CTO技术栈公众号

全部话题 »

编辑推荐

Java agent超详细知识梳理

为 Python 写一个 C++ 扩展模

为什么我觉得GoFrame的Garr
好用?

Spring Boot下如何校验Spring
及如何自定义校验注解

PHP转Go，框架选什么?

相关专题

NVIDIA：数据革命下的前

戴尔
力，
202

Cor
全运
202

我收藏的内容

复制

```
type QuadraticGA struct {}

func (l QuadraticGA) GenerateInitialPopulation(populationSize int) []interface{}{
  initialPopulation := make([]interface{}, 0, populationSize)
  for i:= 0; i < populationSize; i++ {
    initialPopulation = append(initialPopulation, makeNewEntry())
  }
  return initialPopulation
}

func (l QuadraticGA) PerformCrossover(result1, result2 interface{}, _ int) interface{}{
  return (result1.(float64) + result2.(float64)) / 2
}

func (l QuadraticGA) PerformMutation(_ interface{}, _ int) interface{}{
  return makeNewEntry()
}

func (l QuadraticGA) Sort(population []interface{}){
  sort.Slice(population, func(i, j int) bool {
    return calculate(population[i].(float64)) > calculate(population[j].(float64))
  })
}
```

更奇怪的是，我从来没有提到过这些方法的接口。请记住，因为没有对象，也没有继承。QuadraticGA结构体是一个空白对象，隐式地作为GeneticAlgorithmRunner。每个必需的方法都在括号中绑定到该结构体，就像Java中的“@override”。现在，结构体和设置需要传递给运行该算法的模块。

复制

```
settings := ga.GeneticAlgorithmSettings{
   PopulationSize: 5,
   MutationRate: 10,
   CrossoverRate: 100,
   NumGenerations: 20,
   KeepBestAcrossPopulation: true,
}

best, err := ga.Run(QuadraticGA{}, settings)

if err != nil {
   println(err)
}else{
   fmt.Printf("Best: x: %f  y: %f\n", best, calculate(best.(float64)))
}
```

很简单,对吧?“QuadraticGA {}”只是简单地创建了该结构的一个新实例，其余的则由Run()方法完成。该方法返回搜索结果和发生的任何错误，因为Go不相信try / catch——另一场战争作者采取了严格的设计立场。

```
func makeNewEntry() float64 {
   return highRange * rand.Float64()
}

func calculate(x float64) float64 {
   return  math.Pow(x, 2) - 6*x + 2 // minimum should be at x=3
}
```

既然已经为二次实现创建了接口，那么GA本身需要完成:

```
func Run(geneticAlgoRunner GeneticAlgorithmRunner, settings GeneticAlgorithmSettings) (interface{}, e

   population := geneticAlgoRunner.GenerateInitialPopulation(settings.PopulationSize)

   geneticAlgoRunner.Sort(population)

   bestSoFar := population[len(population) - 1]

   for i:= 0; i < settings.NumGenerations; i++ {

      newPopulation := make([]interface{}, 0, settings.PopulationSize)

      if settings.KeepBestAcrossPopulation {
         newPopulation = append(newPopulation, bestSoFar)
      }

      // perform crossovers with random selection
      probabilisticListOfPerformers := createStochasticProbableListOfIndividuals(population)

      newPopIndex := 0
      if settings.KeepBestAcrossPopulation{
         newPopIndex = 1
      }
      for ; newPopIndex < settings.PopulationSize; newPopIndex++ {
         indexSelection1 := rand.Int() % len(probabilisticListOfPerformers)
         indexSelection2 := rand.Int() % len(probabilisticListOfPerformers)

         // crossover
         newIndividual := geneticAlgoRunner.PerformCrossover(
            probabilisticListOfPerformers[indexSelection1],
            probabilisticListOfPerformers[indexSelection2], settings.CrossoverRate)

         // mutate
         if rand.Intn(101) < settings.MutationRate {
            newIndividual = geneticAlgoRunner.PerformMutation(newIndividual)
         }

         newPopulation = append(newPopulation, newIndividual)
      }
```

```
45.
46.         // keep the best so far
47.         bestSoFar = population[len(population) - 1]
48.
49.     }
50.
51.     return bestSoFar, nil
52. }
53.
54. func createStochasticProbableListOfIndividuals(population []interface{}) []interface{} {
55.
56.     totalCount, populationLength:= 0, len(population)
57.     for j:= 0; j < populationLength; j++ {
58.         totalCount += j
59.     }
60.
61.     probableIndividuals := make([]interface{}, 0, totalCount)
62.     for index, individual := range population {
63.         for i:= 0; i < index; i++{
64.             probableIndividuals = append(probableIndividuals, individual)
65.         }
66.     }
67.
68.     return probableIndividuals
69. }
```

很像以前，一个新的人口被创造出来，人口的成员将会世代交配，而他们的后代可能携带突变。一个人的表现越好，就越有可能交配。随着时间的推移，算法收敛到***的答案，或者至少是一个相当不错的答案。

那么当它运行时，它返回了什么呢?

```
1.  Best: x: 3.072833 y: -6.994695
```

复制

不坏!由于人口规模只有5、20代，而且输入的范围被限制在[0 100]，这一搜索就钉在了顶点上。

现在，您可能想知道为什么我定义了所有的接口方法来返回“接口{}”。这就像Go和generics一样。没有对象，因此没有对象类型返回，但是没有描述的大小的数据仍然可以在堆栈上传递。这本质上也是这个返回类型的含义:它传递一些已知的和类似的类型的对象。有了这个“泛型”，我就可以将GA移动到它自己的包中，并将相同的代码移到多个不同类型的数据上。

我们有两个输入的3D二次方程，而不是一个二维二次方程的单个输入。接口方法只需要很小的改变:

```
1.  type Quad3D struct {
2.      x, y float64
3.  }
4.  func makeNewQuadEntry(newX, newY float64) Quad3D {
5.      return Quad3D{
6.          x: newX,
7.          y: newY,
8.      }
9.
```

复制

```
14.  }
15.
16.  type Quadratic3dGA struct {
17.  }
18.
19.  func (l Quadratic3dGA) GenerateInitialPopulation(populationSize int)[]interface{}{
20.
21.     initialPopulation := make([]interface{}, 0, populationSize)
22.     for i:= 0; i < populationSize; i++ { initialPopulation = append(initialPopulation, makeNewQuadEntr
23.     })
24.  }
25.
26.  func quadratic3dMain(){
27.     settings := ga.GeneticAlgorithmSettings{
28.        PopulationSize: 25,
29.        MutationRate: 10,
30.        CrossoverRate: 100,
31.        NumGenerations: 20,
32.        KeepBestAcrossPopulation: true,
33.     }
34.
35.     best, err := ga.Run(Quadratic3dGA{}, settings)
36.     entry := best.(Quad3D)
37.
38.     if err != nil {
39.        println(err)
40.     }else{
41.        fmt.Printf("Best: x: %f  y: %f  z: %f\n", entry.x, entry.y, calculate3D(entry))
42.     }
     }
```

而不是到处都是float64s，任何地方都可以通过Quad3D的条目;每一个都有一个X和一个Y值。对于创建的每个条目，都使用contructor makeNewQuadEntry创建。Run()方法中的代码都没有更改。

当它运行时，我们得到这个输出:

```
1.   Best: x: 3.891671 y: 4.554884 z: -12.787259
```

复制

很接近了!

哦，我忘了就是快了!在Java中执行此操作时，即使使用相同的设置，也会有明显的等待时间。在一个相对较小的范围内求解二次方程并不是很复杂，但它对一个人来说是值得注意的。

Go是本地编译的，比如C。当二进制执行时，它似乎马上就吐出一个答案。这里有一个简单的方法来度量每次运行的执行时间:

```
1.   func main() {
2.      beforeQuadTime := time.Now()
3.      quadraticMain()
4.      afterQuadTime := time.Since(beforeQuadTime)
```

复制

```
9.      after3dQuatTime := time.Since(before3dQuadTime)
10.     fmt.Printf("%d\n", after3dQuatTime)
11. }
```

边注:我能说我很高兴我们是一个开发者社区，让他们从过去的错误中走出来，并把综合的时间模块和包构建成一种语言吗?Java 8 +拥有它们，Python拥有它们，并拥有它们。这使我开心。

现在的输出:

复制

```
1.  Best: x: 3.072833 y: -6.994695
2.  136,876
3.  Best: x: 3.891671 y: 4.554884 z: -12.787259
4.  4,142,778
```

那“近乎瞬间”的感觉是我想要传达的，现在我们有了很难的数字。136,876看起来很大，但要在纳秒内报告时间。

重申一遍:纳秒。不是几毫秒，我们都习惯了在互联网时代或者其他像Python和Java这样的通用语言;纳秒。1/1,000,000毫秒。

这意味着我们在不到一毫秒的时间里找到了一个使用遗传算法来搜索答案的二次方程的答案。这句话，“该死的瞬间”似乎很合适，不是吗?这包括打印到终端。

那么，要计算更密集的东西呢?在我展示一种寻找好的梦幻足球lineups的方法之前，我在Fanduel上使用。这包括从电子表格中读取数据，制作和过滤lineups，并进行更复杂的交叉和突变。强制寻找***解决方案可能需要超过75,000年(至少使用我当时使用的Python)。

我不需要再检查所有的细节，你可以自己去看代码，但我会在这里显示输出:

复制

```
1.  Best: 121.409960:, $58100
2.  QB: Aaron Rodgers - 23.777778
3.  RB: Latavius Murray - 15.228571
4.  RB: DeMarco Murray - 19.980000
5.  WR: Kelvin Benjamin - 11.800000
6.  WR: Stefon Diggs - 14.312500
7.  WR: Alshon Jeffery - 9.888889
8.  TE: Connor Hamlett - 8.200000
9.  D: Philadelphia Eagles - 10.777778
10. K: Phil Dawson - 7.444444
11. 16,010,182
```

哦,是的!现在看来这是一个很好的阵容!它只需要16毫秒就能找到。

现在，这个遗传算法可以改进了。与C一样，当将对象传递给方法时，将在堆栈上复制对象(读取数据)。随着对象大小的增长，***不要反复复制它们，而是要在堆中创建它们，并在周围传递指针。目前，我将把它作为未来的工作。

Go也被用coroutines和信道的原生支持编写，利用多个内核来解决一个问题，比过去简单多了，相比于单核时代的其他语言来说，这是一个巨大的优势。我想要增强这个算法来使用这些工具，但这也必须留给以后的工作。

我很享受学习的过程。对于我来说，用组合而不是继承来考虑工程解决方案是很困难的，因为我已经习惯了8年以上的时间，也是我学会编程的方式。但是每种语言和方式都有各自的优点和缺点;每一种语言在我的工具中都是不同的工具。对于任何担心尝试的人，不要。有一个驼峰(更像是一个减速带)，但你很快就会克服它，走上成功之路。

同话题下的热门内容

Java 循环语句的简要指南

尽管我与一些开发人员的观点存在分歧，以及我必须考虑解决问题的不同方式，但Go真的是一种很好的语言。我鼓励大家在学习一两门语言后再试一试。它很快就成为了***的语言之一，有很多原因可以解释为什么。我期待着在未来更多地使用它。

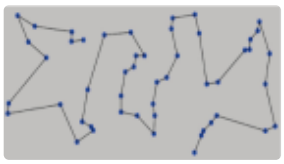
责任编辑：未丽燕 来源： CSDN

Go语言 算法 代码

赞

分享到微信 分享到微博

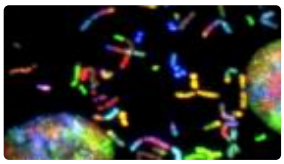
相关推荐



人工智能算法：遗传算法

遗传算法是一种特殊的演化算法，但是在描述遗传算法的文献中，其定义各不相同。本书将遗传算法定义为一种可以用交叉和突变算子优化固定长度向量的演化算法。计分函数可以区分优劣方案，

2021-03-10 15:49:20 人工智能 遗传算法



用Python从头开始实现简单遗传算法

遗传算法是模仿自然选择过程的优化算法。他们没有使用"数学技巧"，而是仅复制了我们知道其有效的逻辑。在本文中，我将更详细地介绍如何理解遗传算法的不同部分。

2020-06-11 08:32:50 Python 遗传算法 代码

用Python从零开始实现简单遗传算法

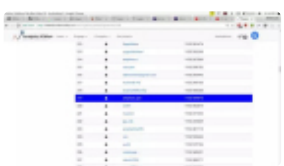
遗传算法是一种随机全局优化算法。连同人工神经网络，它可能是最流行和广为人知的生物学启发算法之一。在本教程中，您将发现遗传算法优化算法。

2021-03-16 11:30:33 Python 遗传算法 神经网络

遗传算法中几种不同选择算子及Python实现

遗传算法(geneticalgorithms,GAs)是一种自适应的启发式搜索算法,它模仿达尔文进化论中的“适者生存”的原则,最终获取优化目标的最优解。

2017-09-22 15:03:08 Python 遗传算法 GAFT框架



一文读懂遗传算法工作原理（附Python实现）

本文是作者ShubhamJain现身说法，用通俗易懂的语言对遗传算法作了一个全面而扼要的概述，并列举了其在多个领域的实际应用，其中重点介绍了遗传算法的数据科学应用。

2017-08-21 10:00:23 遗传算法 Python 生物学

GAFT：一个使用Python实现的遗传算法框架

对于遗传算法来说，就非常适合写个相对固定的框架然后给算子、参数等留出空间以便对新算法进行测试和改进。于是就动手写了个遗传算法的小框架gaft，本文对此框架进行一些介绍并分别以一

同话题下的热门内容

Java 循环语句的简要指南

51CTO技术栈公众号

基因遗传算法是一种灵感源于达尔文自然进化理论的启发式搜索算法。该算法反映了自然选择的过程，即最适者被选定繁殖，并产生下一代。本文简要地介绍了遗传算法的基本概念和实现，希望能

2017-07-12 14:23:25 遗传算法 java 自然选择

C#遗传算法学习笔记

本文介绍C遗传算法学习笔记，通过运行程序，你会发现通过不断的进化，种群的总的适应环境的能力在逐步提高。

2009-08-14 09:41:03 C#遗传算法

机器学习算法实践-Platt SMO和遗传算法优化SVM

本文在之前简化版SMO算法的基础上实现了使用启发式选取α对的方式的PlattSMO算法来优化SVM。另外由于最近自己也实现了一个遗传算法框架GAFT，便也尝试使用遗传算法对于SVM的原始

2017-10-17 14:25:56 机器学习 算法 优化

在Python中用遗传算法优化垃圾收集策略

在本文中，我将展示如何在Python中实现一个遗传算法，在几个小时内“进化”一个收集垃圾的机器人。

2020-10-26 13:42:28 Python 算法 垃圾

IT生活：用遗传算法让电脑为你写宋词

相逢缥缈,窗外又拂晓.长忆清弦弄浅笑,只恨人间花少.菊不待清尊,相思飘落无痕.风雨重阳又过,登高多少黄昏.

2010-05-11 11:00:44 遗传算法 宋词

Go 语言实现常见排序算法

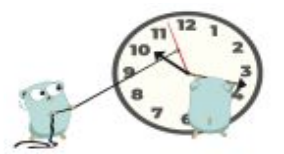
利用递归与分治技术将数据序列划分成为越来越小的半子表，再对半子表排序，最后再用递归步骤将排好序的半子表合并成为越来越大的有序序列。其中“归”代表的是递归的意思，即递归地将数组

2022-11-01 18:29:25 Go 语言 排序算法

基于遗传算法的高频标签天线的优化设计

本文对标签天线的集成化技术进行研究，并且采用遗传算法对片上天线的几何参数和工艺参数进行优化，快速得到满足性能要求的天线尺寸。仿真结果表明，在工艺方法确定的情况下，采用此优化算法得到的天线可以更好地提高电路的工作效率。并且，如果工艺方法可选，

2014-11-28 16:08:33 射频识别 RFID



Go语言如何实现stop the world?

在某些垃圾回收器算法中，“停止世界”(StoptheWorld:STW，下同)是跟踪内存使用最重要的阶段，它会停止程序的执行，以扫描内存使用，并添加写障碍。让我们回顾一下它在内部如何工作，

2020-03-17 10:24:12 Go语言 停止 写障碍

几种限流算法的Go语言实现

不依赖外部库的情况下，限流算法有什么实现的思路？本文介绍了3种实现限流的方式。

2022-05-19 14:14:26 go 语言 限流算法

AI系统首次实现真正自主编程：利用遗传算法，完爆初级程序员

同话题下的热门内容

Java 循环语句的简要指南

51CTO技术栈公众号

使用 Go 语言实现汉诺塔（Hanota）算法

我最近重温了一下《猩球崛起》这部电影。在电影中，凯撒就玩了河内塔游戏。你还有印象吗？其实独自一人玩一些游戏是好难的😭（译者不知作者为何这么说🤔，难道是无聊嘛？🤓），今天我

2022-04-18 10:01:07 Go 语言 汉诺塔 游戏

AI编程完爆原作者，程序员：我编写的程序让自己失业了

彭博和英特尔实验室的两位研究人员，利用遗传算法和图灵完备语言，实现了首个能够自动编程的AI系统“AIProgrammer”。

2017-10-27 18:20:59 程序员

软考程序员：标准的遗传算法求函数最大值

以下是软考程序员：标准的遗传算法求函数最大值。

2011-01-19 11:14:45 程序员

Go 语言如何实现字符串切片反转函数

本文通过Python中的reverse()函数的一个示例，引发出一个思考：Go语言中有没有类似的反转函数?

2022-11-10 07:43:45

同话题下的热门内容

Java 循环语句的简要指南

51CTO技术栈公众号