

Evolutionäre Algorithmen - WS2012/13

Übungsblatt 7 - Ökolopoly

Sebastian Nuck

22. Januar 2013

Inhaltsverzeichnis

1	Konzept	2
1.1	Erstellung von Startkonfigurationen	2
1.2	Evolutionärer Algorithmus	2
2	Repräsentation	3
3	Mutation, Selektion und Gütebewertung	4
3.1	Mutation	4
3.2	Selektion	4
3.3	Gütebewertung	4
4	Simulationsszenarien und Effizienz	5
5	Fazit	6
	Literaturverzeichnis	7

1 Konzept

Als Basis des Gesamtkonzepts dient ein Neuronales Netz. Um das Netz zu trainieren, mutiert der Algorithmus die Gewichte und Schwellwerte des Netzes. Damit es zu keine Übertrainierung kommt, wird mit verschiedenen Startkonfigurationen für die Klasse Kybernetien gearbeitet.

1.1 Erstellung von Startkonfigurationen

Um eine Menge von unterschiedlichen Konfigurationen für den Simulator zu erstellen, wird die Datenstruktur `KypInputs` verwendet. Diese beinhaltet alle Parameter, die der Simulator zum Start benötigt. Ein dafür implementierter Generator erstellt verschiedene Konfigurationen mit zufälligen Parametern in einem vorher definierten Bereich und stellt diese als Liste für den Evolutionären Algorithmus bereit.

1.2 Evolutionärer Algorithmus

Der Algorithmus erstellt anfangs eine zufällige Startbelegung für die Bestandteile des neuronalen Netzes. Diese wird im Laufe des Algorithmus mutiert. Damit wird versucht, das Netz möglichst gut auf die generierten Startkonfigurationen anzulernen.

2 Repräsentation

Die Strategie wird als neuronales Netz mit folgenden Eigenschaften dargestellt.

- Eingabeneuronen: 9
- Hidden-Layer: 8
- Neuronen je Hidden-Layer: 20;
- Ausgabeneuronen: 6

Die Anzahl der Eingabeneuronen wird durch die 9 Eingabewerte bestimmt. Die 6 Ausgabeneuronen kommen durch die 5 aktiv beeinflussbaren Werte sowie durch die zusätzliche Option, Aufklärungspunkte für oder gegen Bevölkerung zu investieren, zustande.

Die Anzahl der Hidden-Layer und deren Neuronen wurde durch ständiges optimieren und testen ermittelt. Dabei erwiesen sich diese Anzahlen als optimal für das Problem. Mehr Hidden-Layer und Neuronen führten nur zu einer Laufzeitverlängerung und hatten keine Verbesserung der Güte zur Folge. Weniger Layer oder Neuronen pro Layer erwiesen sich im Gegensatz dazu als nicht geeignet, derart viele Parameter zu optimieren.

Ein Neuron wird innerhalb des Individuums wie in Abbildung 2.1 dargestellt.

Neuron
List<Neuron> inputs; double weight; double threshold; double value;
«constructor» -Neuron(double weight, double threshold, List<Neuron> inputs); «misc» calc();

Abbildung 2.1: Die Klasse Neuron

Die Verknüpfung der Neuronen untereinander wird mit Hilfe des Konstruktors realisiert. Damit wird jedem Neuron eine Liste von eingehenden Neuronen übergeben. Dann wird die Methode `calc()` aufgerufen. Dort werden die Gewichte und Schwellwerte aufsummiert und mit Hilfe der Sigmoidfunktion der Wert des Neurons berechnet. Für alle Parameter außer *Produktion* und der Option, Aufklärung für oder gegen die Bevölkerung zu investieren wird die Sigmoidfunktion wie in Formel 2.1 verwendet. Für die beiden anderen Parameter wird eine Funktion benötigt, die Werte von -1 bis $+1$ liefert, um eine positive oder negative Investition darzustellen. Dafür wird der Tangens Hyperbolicus wie in Formel 2.2 verwendet.

$$sig(x) = \frac{1}{1 + e^{-x}} \quad (2.1)$$

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (2.2)$$

3 Mutation, Selektion und Gütebewertung

3.1 Mutation

Als Basis für die Mutation dient die `SELBSTADAPTIVE-EP-MUTATION` wie im Algorithmus 4.19 in [Wei07] beschrieben. Im Detail werden die Gewichte, Schwellwerte und deren beider Strategieparameter mutiert, indem ein bestimmter Anteil eines zufällig gewählten Wertes aufaddiert wird.

Für die Bestimmung der gaußverteilten Zufallszahlen wird nicht die Standardimplementierung von Java verwendet sondern eine in Java geschriebene Version des `Mersenne Twister`. Die Implementierung stammt von *Sean Luke* [Luk]. Dieser Generator wurde der Standardimplementierung vorgezogen, da er ca. $\frac{1}{3}$ schneller ist als Java's `Random`.

Diese Art der Mutation wurde verwendet, weil sie eine Anpassung der Parameter des Netzes erlaubt. Dabei werden bei jeder Mutation alle der 100 Werte der Startpopulation mutiert. Als Resultat werden dieser Population die 100 mutierten Individuen angehängt.

3.2 Selektion

Als Selektion wird eine Bestenselektion verwendet. Dabei werden die besten 100 der insgesamt 200 Individuen entnommen. Diese stellen die Startpopulation für den neuen Zyklus des Algorithmus dar. Die Auswahl dieser Selektion lässt sich damit begründen, dass für das Lernen des Netzes möglichst nur die Besten verwendet werden sollen. Durch die Durchmischung mit den nicht-mutierten Individuen wird garantiert, dass mutierte Individuen, die schlechtere Werte liefern als vor der Mutation sich erneut in der Startpopulation befinden.

Eine bessere Diversität der Individuen hätte sich eventuell durch die `Q-STUFIGE-TURNIER-SELEKTION` nach Algorithmus 3.7 aus [Wei07] erreichen lassen können. Allerdings wurde dieser Ansatz nicht weiter verfolgt.

3.3 Gütebewertung

Die Güte eines Individuum wird im Algorithmus durch die Anzahl der überlebten Zyklen bestimmt. Dabei wird nicht allein die Standardkonfiguration verwendet. Vielmehr werden mithilfe des Generators eine Menge unterschiedlicher Konfigurationen erstellt dessen mittlere Güte für die Bewertung der Individuen verwendet wird.

4 Simulationsszenarien und Effizienz

Für ein bestmögliches Training des Neuronalen Netzes wird versucht, mithilfe des Generators eine größtmögliche Diversität an Startkonfigurationen für den Simulator bereitzustellen. Als Basis dienen folgende Werte.

Parameter	Wertebereich	Standardparameter
<i>Aktionspunkte</i>	5 ... 11	8
<i>Sanierung</i>	1 ... 7	1
<i>Produktion</i>	9 ... 15	12
<i>Umweltbelastung</i>	5 ... 11	13
<i>Aufklärung</i>	1 ... 7	4
<i>Lebensqualität</i>	7 ... 13	10
<i>Vermehrungsrate</i>	17 ... 23	20
<i>Bevölkerung</i>	18 ... 24	21
<i>Politik</i>	-3 ... 3	0

Tabelle 4.1: Verwendete Wertebereiche zur Bestimmung des besten Individuums

Insgesamt wurden 80 Konfigurationen mit zufälligen Parameter in den Bereichen aus Tabelle 4.1 erstellt. Zusätzlich wurden noch für jeden Parameter Extremwerte, positiv wie negativ, hinzugefügt. Die genauen Werte der verwendeten Konfigurationen können aus dem beiliegenden Logfile entnommen werden.

5 Fazit

Literaturverzeichnis

- [Luk] LUKE, Sean: The Mersenne Twister in Java, URL <http://www.cs.gmu.edu/~sean/research/>, last checked: 22.01.2013
- [Wei07] WEICKER, Karsten: *Evolutionäre Algorithmen (XLeitfäden der Informatik)*, Vieweg+Teubner Verlag, 2., überarb. u. erw. Aufl. 2007 Aufl. (2007), URL <http://amazon.de/o/ASIN/3835102192/>