Evolutionäre Algorithmen - WS2012/13

Übungsblatt 7 - Ökolopoly

Sebastian Nuck

24. Januar 2013

Inhaltsverzeichnis

1 Konzept

Als Basis des Gesamtkonzepts dient ein Neuronales Netz. Um dieses 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.

Eine Besonderheit stellt eine einstellbare Abbruchbedingung dar. Mit deren Hilfe ist es möglich, im Falle von zu vielen erfolglosen Iterationen die Startparameter mit neuen Zufallszahlen zu füllen. Dadurch wird dem Algorithmus ermöglicht, aus eventuell auftretenden lokalen Optima herauszukommen. Auch wenn dies ein sehr zufällig bestimmter Vorgang ist, konnte er sich im Laufe der Tests profilieren und stets eine höhere Anzahl von Runden erreicht werden.

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.

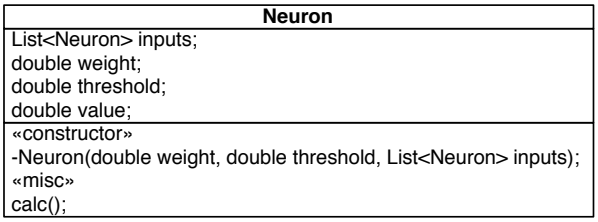

Abbildung 2.1: Die Klasse Neuron

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.

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

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

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 sonder eine in Java geschriebene Version des `MersenneTwister`. 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 nicht 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. Jedes mal, wenn ein neuer besserer Durchschnittswert erreicht wird, wird das Individuum serialisiert.

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. Es wurden 100 Individuen als Startpopulation verwendet. Als Basis für die Startkonfigurationen des Simulators für das abgegebene Individuum dienten die Wertebereiche, wie in Tabelle 4.1 beschrieben.

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

Um sich schließlich auf diese Konfiguration festlegen zu können, mussten vorher viele Tests abgearbeitet werden. Das Netz wurde zu Beginn nur auf die Standardkonfiguration angelernt um die grundsätzliche Funktionalität zu überprüfen. Mit der Verwendung des Generators konnten später verschiedene Konfigurationsbereiche abgedeckt werden. Dazu werden vorher festgelegte Standardwerte um ein einstellbares ε in die positive wie auch in die negative Richtung verschoben. Mithilfe des ε wird dadurch ein Bereich festgelegt aus dem sich der Zufallszahlengenerator einen Wert entnimmt. Für $\varepsilon = 1$ bis $\varepsilon = 2$ konnte selbst bei 500 zufällig gewählten Startkombinationen ein Durchschnitt von 30 Runden erreicht werden. Tabelle 4.2 zeigt, wie viele Runden in max. 120s erreicht werden konnten.

ε	max. Runden	Iterationen	Laufzeit in [s]
0	30	227	2
1	30	1365	85
2	30	1169	90
3	26	750	120
4	25	1600	120
5	25	800	120
6	22	500	120
7	10	900	120

Tabelle 4.2: Maximal erreichte Runden in 120s

5 Fazit

Das geplante Gesamtkonzept mit neuronalem Netz hat sich als durchaus geeignet erwiesen. Nachdem das Netz erst einmal implementiert war konnten schnell gute Werte erreicht werden. Mit der Hilfe des Startkonfiguration-Generators konnte außerdem die nötige Diversität ermittelt und gleichzeitig ein Übertrainieren auf eine bestimmte Konfiguration vermieden werden.

Allerdings wird schnell ersichtlich, dass das ganze Verfahren stark auf Zufallswerten beruht. Äußerst entscheidend ist dabei die Startpopulation. Wurden da geeignete Gewichte und Schwellwerte erzeugt konnte der Algorithmus sehr schnell eine hohe Rundenanzahl erreichen. Dieser Erkenntnis führte auch zum Anwenden der Abbrucheinschaft, die greift, wenn über mehrere Iterationen kein besseres Ergebnis gefunden werden konnte. Durch die neue Generierung der Gewichte und Schwellwerte konnten auch in der Praxis hohe Rundenanzahlen erreicht werden. Mithilfe von Backpropagation könnte man an dieser Stelle ansetzen und noch bessere Ergebnisse erreichen.

Ich denke, dass mit dem erreichten Individuum ein gutes Allround-Individuum erzeugt wurde. Tests zeigten, dass es wenig anfällig gegen Extremwerte ist. Einzig der Wert *Produktion* führt relativ schnell zu einem Sinken der Rundenanzahl.

Ob mit Hilfe dieses implementierten neuronalen Netzes eine bessere Rundenanzahl, oder sogar das Optimum - dauerhaft 30 Runden - erreicht werden kann ist fraglich. Da aufgrund der begrenzten Zeit die Testläufe nach max. 48h für Auswertungen und Neukonfigurationen unterbrochen werden mussten ist außerdem nicht klar, ob das Netz nach einer größeren Zeitdauer irgendwann zu einem optimalen Ergebnis gekommen wäre. Wahrscheinlicher ist, dass es, wie in den stattgefundenen Testläufen, in lokalen Optima hängen bleibt und nur mit einer günstigen Zufallszahlkombination weiterläuft. Vermutlich wird sich das Netz nach einer gewissen Zeit in eine, für die Startparameter geeigneten, Konfiguration übertrainiert haben.

Leider war die angesetzte Zeit für die Bearbeitung des Projektes ungünstig bemessen. Allein für die Einarbeitung in Grundlagen zu Neuronalen Netzen und für die Basisimplementierung mussten min. 20h Arbeitszeit investiert werden. Daraus folgte natürlich, dass für die eigentliche Optimierung des Programms nur noch relativ wenig Zeit zur Verfügung stand. Ich war zwar in der Lage alle geplanten Features zu implementieren, jedoch hätte ich gerne die Zeit gehabt, weitere Selektionsmechanismen oder andere Strategien zu implementieren.

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/