Effiziente Entwicklung neuronaler Netzwerktopologien durch reinforcement learning

Harald Heckmann

Studiengang Angewandte Informatik, Hochschule RheinMain
harald.heckmann@student.hs-rm.de

Abstract

In diesem Dokument wird die grundlegende Arbeitsweise von Neuronalen Netzen erklärt. Aufbauend darauf, wird eine Methode eingeführt, NEAT, welche zur Optimierung neuronaler Netze verwendet wird. Anschließend wird erklärt, wie ein Computerspiel mit neuronalen Netzen, die mit NEAT optimiert wurden, gewonnen werden kann.

I. Einleitung

I.1 Super Mario

Ziel des Spiels Super Mario ist es, die Spielfigur Mario so schnell wie möglich in das Ziel zu führen und auf dem Weg möglichst viele Münzen zu sammeln und Gegner zu zerstören, da diese zusätzlich Punkte geben. Mario ist eine Spielfigur die zu Fuß unterwegs ist und sich ducken, springen und rennen kann. Mario kann Verstärkungen aufsammeln, die sein Durchhaltevermögen erhöhen als auch die Fähigkeit bieten, Projektile abzufeuern, die dazu benutzt werden sich vor Gegnern zu schützen. Mario hat zu Beginn 3 Leben und die Berührung eines Gegners sowie das Fallen in eine Grube können einen Verlust eines Lebens bedeuten. Wurden alle Leben verloren, so ist das Spiel vorbei.

I.2 Einsatz neuronaler Netze

Neuronale Netze werden verwendet, wenn ein System (selbstständig oder mit einem Lehrer) etwas lernen soll. Das Neuronale Netz besteht aus Knoten (Neuronen) und Kanten (Synapsen). Dabei ist interessant, wie viele Schichten (Ebenen) mit Knoten es gibt und in welche Richtung die Knoten über die Kanten Signale senden können sowie diese gewichtet sind. Je nach Anzahl der Ebenen und der Flussrichtung der Signale können verschiedene Anwendungsfälle abgedeckt werden. Die Gewichtung spielt eine maßgebliche Rolle, da diese entscheidend dafür ist, in welcher Reihenfolge Reize ausgeführt werden. Dafür werden Lernregeln gebraucht.

I.3 Evolutionäre Algorithmen

Evolutionären Algorithmen liegen die darwinischen Prinzipien als Grundsatz vor. Diese Algorithmen gehören von der Arbeitsweise her üblicherweise zu der Kategorie "trial and error" (Versuch und Irrtum, Ausprobieren). Dabei werden metaheuristische oder stochastische Methoden angewandt um eine Optimierung zu erreichen. Neue neuronale Netze entstehen durch die Kombination oder Mutation bestehender neuronaler Netze. Neuronale Netze können auch verworfen werden, wenn ihre Fitness zu gering ist. Die Fitness spiegelt den Erfolg hinsichtlich einer bestimmten Aufgabe eines neuronales Netzes wieder.
Beispiel: In einem Rennspiel ist die Fitness höher desto schneller man ins Ziel kommt.

I.3.1 NEAT

NEAT ist eine Methode die dazu dient die Struktur des neuronalen Netzes als auch die Gewichte der Verbindungen zu optimieren. Dieser Methode liegt ein genetischer Algorithmus zu Grunde, welcher eine Unterkategorie der evolutionären Algorithmen darstellt. Auf diesen Algorithmus wird weiter im Kapitel NEATIII eingegangen. Die Ideen von NEAT lauten:

- Eine Logik zum Verbinden der Knoten (Gene) um neue Lösungswege zu erzeugenIII.2
- Ein Mechanismus zum Schutz der strukturellen InnovationIII.3
- Der Suchraum der Lösungen soll minimal gehalten werdenIII.4

I.4 Reinforcement Learning

Reinforcement learning wird durch ein Lernproblem charakterisiert. Immer wenn das System auf Rückmeldung der Umgebung angewiesen ist, ist Reinforcement Learning angebracht. Beispiel Super Mario: Das neuronale Netz hat sich so geformt, dass Mario springt wenn ein unbekanntes Objekt auf Mario zukommt. Ein neuer Gegner erscheint und das Springen Marios führt zum Verlust eines Lebens, woraufhin das neuronale Netz so umgeformt wird, dass ein Sprung vermieden wird wenn der gleiche Gegner auf Mario zukommt.

II. Neuronale Netze

II.1 Grundlagen

II.1.1 Units (nodes)

Alle hier erläuterten Informationen über neuronale Netze wurden aus den Quellen [1],[2] und[3] erlernt.
Das neuronale Netz kann mit Hilfe der Graphentheorie beschrieben werden. Dabei sind die Units, auch Neuronen genannt, die Knoten. Man unterscheidet zwischen drei verschiedenen Arten von Neuronen:

- Input-Neuronen - Neuronen die von der Außenwelt Signale empfangen können
- Output-Neuronen - Neuronen die Signale an die Außenwelt weitergeben
- Hidden-Neuronen - Neuronen, die sich zwischen Input- und Outputneuronen befinden

II.1.2 Connections

Die Connections, auch Synapsen genannt, sind die Verbindungen zwischen den Neuronen, die Kanten. Diese sind gerichtet und mit einem Gewicht versehen. Dieses Gewicht gibt an, wie stark zwei Neuronen aufeinander wirken

1. positives Gewicht - das Neuron hat einen verstärkenden Einfluss auf das andere Neuron am Ende der Kante (in Richtung der Kante)
2. negatives Gewicht - das Neuron hat einen hemmenden Einfluss auf das andere Neuron am Ende der Kante (in Richtung der Kante)
3. Gewicht von Null - die beiden Neuronen haben keine Wirkung zueinander

Sämtliches Wissen des Neuronalen Netzes ist in seinen Gewichten gespeichert.

II.1.3 Propagierungs- und Aktivierungsfunktion

Der Netzinput fasst den gesamten Input eines Neurons zusammen. Dieser wird mittels der Propagierungsfunktion errechnet. Der daraus resultierende Netzinput ist maßgeblich um anschließend den neuen Aktivierungszustands des (empfangenden) Neurons mittels der Aktivierungsfunktion zu bestimmen.

Die Standardpropagierungsfunktion summiert alle Inputs auf:

$$netinput_i = \sum_j input_{ij} = \sum_j a_j \cdot w_{ij}$$

$input_{ij}$ = Input von Neuron j nach Neuron i
a_j = Aktivitätslevel des sendenden Neurons
w_{ij} = Gewicht der Kante zwischen Neuron i und Neuron j

Anschließend wird der Aktivierungszustand des Neurons i (empfangendes Neuron) mittels einer Aktivierungsfunktion bestimmt:

$$output_i = f_a(netinput_i)$$

meist wird diesem noch ein Schwellwert w_0 zugeordnet, sodass:

$$output_i = f_a(netinput_i - w_0)$$

Ein Beispiel einer Aktivierungsfunktion ist die logistische Funktion:

$$f_a(x) = \frac{1}{1 + e^{-x}}$$

II.2 Lernen

II.2.1 Trainings- und Testphase

In der Trainingsphase lernt das Neuronale Netz mit Trainingsmaterial. Hierbei werden die Gewichte der Synapsen angepasst. Wie diese Gewichte berechnet werden, wird im nächsten Kapitel "Lernregeln" erläutert. Das Lernen kann auf zwei Arten vollzogen werden:

- supervised - das gewünschte Ergebnis ist bekannt
- unsupervised - das gewünschte Ergebnis ist nicht bekannt (hier kann NEAT verwendet werden)

nach der Trainingsphase folgt die Testphase. Den Inputneuronen werden Reize übermittelt und die Reaktion des Systems darauf wird geprüft. Folgende Reize gilt es zu unterscheiden:

- Ausgangsreize - Hierbei ist von Interesse, ob das neuronale Netz das Trainingsmaterial richtig erfasst hat
- neue Reize - Hierbei ist von Interesse, ob das Neuronale Netz darüber hinaus in der Lage gewesen ist eine Generalisierung des Trainingmaterials zu vollziehen

II.2.2 Lernregeln

Um die Gewichte der Synapsen anzupassen braucht man Regeln. Die Gewichtsanpassung einer Synapse hängt stark von der Aktivität der Synapse ab. Es gibt verschieden Lernregeln für neuronale Netze, die bestimmen welches Aktivitätslevel Synapsen haben. Dies geschieht unabhängig von NEAT, bei der Optimierung des neuronalen Netzes durch NEAT werden jedoch auch Gewichte gesetzt, was im weiteren Verlauf dieses Papers genauer erläutert wird.
Die Hebb-Regel ist eine Lernregel und wird im weiteren Verlauf als einfaches Beispiel verwendet. Die Hebb-Regel arbeitet nach folgendem Grundsatz: Wenn beide Neuronen gleichzeitig aktiv sind, wird das Gewicht zwischen ihnen verändert. Dies geschieht nach folgender Formel:

$$\triangle w_{ij} = a_i \cdot a_j \cdot \epsilon$$

$\triangle w_{ij}$ = Gewichtsveränderung der Synapse zwischen Neuron i und j
a_i = Aktivitätslevel des empfangenden Neurons
a_j = Aktivitätslevel des sendenden Neurons
ϵ = positiver Lernparamter

III. NEAT

Diesem Kapitel liegen die Quellen [4] und [5] zu Grunde.
NEAT verwendet einen genetischen Algorithmus. Genetische Algorithmen sind spezielle Evolutionäre Algorithmen, welche Lösungen zu Optimierungsproblemen liefern. Die Arbeitsweise der Algorithmen ist geprägt durch die natürliche Evolution. Die neuronale Netze können durch Vererbung, Selektion, Kreuzung und Mutation geformt werden. NEAT kann sowohl die Gewichte der Synapsen als auch die Netzwerkstruktur ändern. Der Vorteil von NEAT ist, dass neuronale Netze bei ihrer Entwicklung geschützt werden und nicht auf Grund von einer zu geringen Fitness zu früh verworfen werden, sodass deren Nutzen für den gegebenen Anwendungsfall besser ermittelt werden kann, bevor darüber bestimmt wird ob das neuronale Netz verworfen oder behalten wird. Des Weiteren werden die neuronalen Netze (nach Start mit einer minimalen Struktur) minimal gehalten, sodass der Suchraum der Lösungen auch minimal verbleibt und Lösungen effizienter gefunden werden können.

Pseudocode: Eine Generation in NEAT

```
Erstelle Menge aus einfachen
Zufallsgenomen

Solange das Programm laeuft {
    Fuehre Trainingsphase einige
    Male aus

    ------- Trainingsphase -------
    Fuehre Kantenmutationen fuer
    x% Zufallsknoten in
    jedem Genom durch

    Fuehre Knotenmutationen fuer
    y% Zufallskanten in
    jedem Genom durch

    # Kein Teil aus NEAT:
    Trainiere die Gewichte der
    Kanten, z.B. mit der Hebb-Regel

    ------- Testphase -------
    Evaluiere Fitness fuer
    alle Genome

    # Den folgenden Teil kann man
    # nach Testphase oder vor
    # Trainingsphase durchfuehren
    Ordne alle Genome
    einer Spezies zu

    Bestimme fuer alle Genome
    die angepasste Fitness

    Bestimme die durchschnittliche
    angepasste Fitness aller Genome

    Bestimme mit den errechneten
    Werten die neuen
    Speziengroessen

    Kreuze und loesche Genome, so
    dass die vorgegebene Spezien-
    Groesse erreicht wird

    inkrementiere den
    Generationszaehler
}
```

III.1 Struktur

Ein durch NEAT entstandenes und optimiertes neuronales Netz hat Genotype (Genome) und Phenotype (Netzwerke). Ein Genom ist ein neuronales Netz. Die Genome beinhalten alle Informationen die erlernt wurden, die Phenotype sind lediglich (eine baumartige) Darstellung dieser Genome.

III.1.1 Genome (Genotyp) und Gene

ein Genom ist eine Struktur die sich aus zwei weiteren Strukturen zusammensetzt:

- Knotengene (node genes) - Neuronen
- Kantengene (connection genes) - Synapsen

Ersteres enthält Informationen bezüglich des Types des Genes (z.B. Sensor, Output). Zweiteres enthält Informationen über die Verbindung dieser Gene. Eine Graphisches Beispiel für ein Genom ist in Abbildung 1 enthalten.

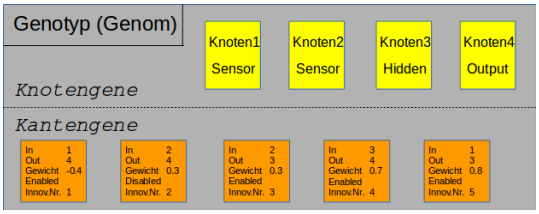

Abbildung 1: Genotyp (Genom)

III.1.2 Netzwerk (Phenotyp)

Möchte man das Genom aus Abbildung 1 als Phenotyp darstellen, so würde das folgendermaßen aussehen:

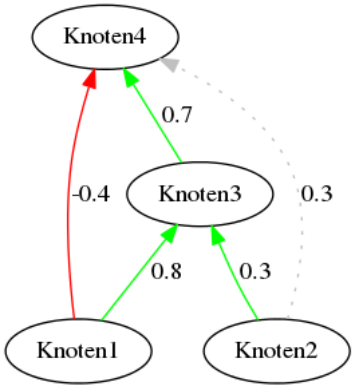

Abbildung 2: einziges Netzwerk zum Genom aus Abb. 1

III.1.3 Genetische Enkodierung (Informationen der Gene)

Eine Kante enthält folgende Informationen:

- In: Knoten, an dem die Verbindung anfängt (Verbindungen haben also Richtungen)
- Out: Knoten, an dem die Verbindung aufhört
- Gewicht: Gewichtung der Verbindung, welche die Wichtigkeit der Verbindung ausdrückt
- Enabled/Disabled: Aktiv / Inaktiv
- Innov.Nr.: Innovationsnummer um die Historie zu verfolgen, dazu später mehr

III.2 Weiterentwicklung der Struktur

Es gibt zwei Möglichkeiten, wie ein Genom mutiert:

1. "Add connection mutation"
   - Zwei bisher unverbundene Knotengene werden verbunden
   - Eine neue Verbindung entsteht
   - Das Gewicht der Verbindung wird zufällig gewählt
2. "Add node mutation"
   - Ein Knoten (Gen) wird zwischen einer bestehenden Verbindung eingefügt
   - Die betroffene Verbindung wird deaktiviert und es entstehen zwei neue Verbindungen
   - Die Kante die zum neuen Knoten führt erhält ein Gewicht von 1.0
   - Die Kante die vom neuen Knoten wegführt erhält das Gewicht der Kante die zuvor an dieser Stelle war und aufgeteilt wurde

Eine graphische Darstellung der Mutationen ist in Abbildung 3 enthalten.

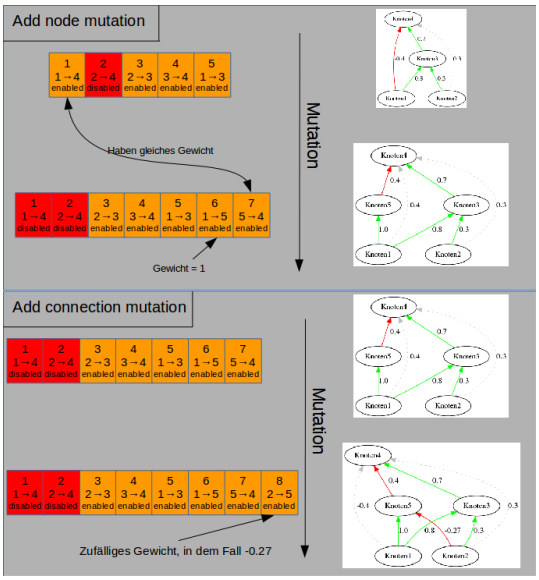

Abbildung 3: Mögliche Mutationen eines Genoms

III.2.1 Historie der Gene

Um zu wissen, welche Gene gut zusammenpassen muss lediglich die Historie festgehalten werden. Zwei Gene aus topologisch verschiedenen Netzwerken müssen, bei gleicher Historie, die gleiche Struktur repräsentieren (die Gewichtung kann sich unterscheiden). Das ist so, da die Strukturen sich aus dem selben Vorfahren ergeben haben. Immer wenn ein neues Gen erzeugt wird, wird diesem eine globale Innovationsnummer zugewiesen, welche anschließend inkrementiert wird. Diese Innovationsnummer repräsentiert eine chronologische Abfolge des Erscheinen der Gene. Wenn Genome sich paaren, wird die Innovationsnummer für jedes Gen vererbt (Historie bleibt erhalten).

III.2.2 Kreuzung der Genome

Im folgenden wird erklärt welche Paarungszustände zwei Genome haben. Diese Paarungszustände werden dazu verwendet herauszufinden, welche Gene bei der Kreuzung zweier Genome für das neu entstehende Genom verwendet werden. Des Weiteren werden die Paarungszustände verwendet, um zu entscheiden, in welche Spezies ein Genom gehört (siehe Abschnitt III.3.1). Wenn zwei Genome miteinander verglichen werden, können die Gene folgende Paarungszustände haben:

1. matching genes (M) - beide Gene aus den beiden zu paarenden Genomen haben die selbe Innovationsnummer

2. disjoint genes (D) - Gene, die keinen Partner und eine Innovationsnummer kleiner als die größte Innovationsnummer des kleineren Genomes haben (passiert üblicherweise bei der "Add Node Mutation")

3. excess genes (E) - Gene, die keinen Partner und eine Innovationsnummer größer als die größte Innovationsnummer des kleineren Genomes haben

   Die zu vererbenden matching genes werden zufällig aus einem der beiden Genome gewählt. Für disjoint und excess genes gilt das auch, wenn beide Genome gleich Fit sind (das bedeutet, dass alle disjoint und excess genes übernommen werden). Sollte das nicht der Fall sein, werden disjoint und excess genes vom fitteren Genom vererbt.
   Anmerkung: Gleich Fit kann auch bedeuten, dass die Fitness beider Genome sich maximal um den Faktor 0.1 unterscheiden darf.
   Abbildung 4 enthält eine graphische Darstellung der Kreuzung zweier Genome.

   Durch die Kreuzung kann das System (wie bei Mutationen auch) eine Population mit unterschiedlichen Topologien erzeugen, wodurch sich ein weiteres Problem ergibt: Das neuronale Netz, dessen Struktur verändert wurde oder erst entstanden ist, kann selber nicht die topologischen Innovationen aufrecht erhalten. Üblicherweise ist eine Folge einer Strukturänderung eines Genoms, dass die Fitness des Genoms sinkt. Das liegt daran,

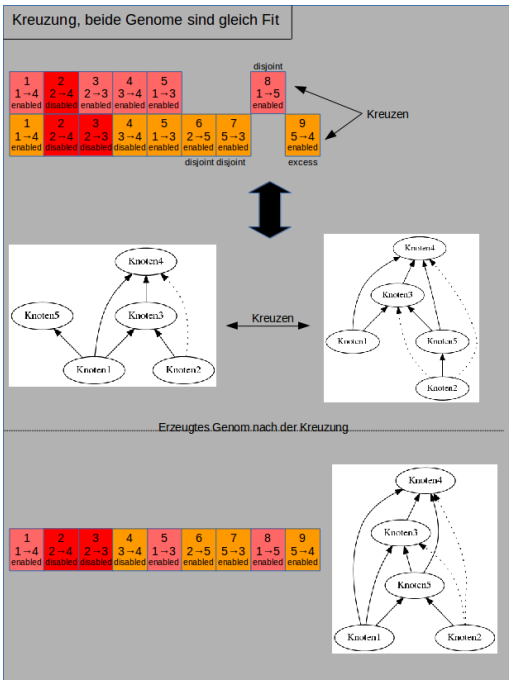

Abbildung 4: Kreuzung zweier Genome mit gleicher Fitness

dass erstens die Strukturänderung nicht unbedingt eine Verbesserung verspricht (und man sich somit vom angestrebten Ziel entfernt) und zweitens die Gewichtung der entstandenen Kante(n) zunächst ermittelt werden muss, bevor diese Sinnvoll verwendet wird/werden.

III.3 Schutz vor genetischer Verarmung durch Spezienbildung

Damit Genome nicht

1. wegen zu geringer Fitness zu früh entfernt werden

2. durch zu rasantes Wachstum die gesamte Population übernehmen

werden Methoden aus den folgenden zwei Kapiteln in NEAT angewandt.

III.3.1 Bestimmung der Mitglieder einer Spezies

Damit Genome bei rasantem Wachstum (der Anzahl der Knoten und Kanten darin) nicht sofort entfernt werden, werden Genome in Spezien eingeteilt. Die Spezienzuweisung erfolgt folgendermaßen: Genom A soll in eine Spezies eingeordnet werden. Zunächst wird Genom B zufällig aus einer Spezies ausgewählt. die Distanz beider Genome δ wird bestimmt und wenn $\delta < \delta_s$ gilt, dann wird Genom A der Spezies zugeordnet. Die Spezienzuordnung wird dann für Genom A abgebrochen, damit Genom A nur einer Spezies zugewiesen wird. Sollte $\delta < \delta_s$ nicht gelten, wird die Prozedur für ein zufälliges Genom aus einer anderen Spezies fortgeführt.

Dafür werden folgende Werte benötigt:

- Kompatibilitätsdistanz δ zwischen zwei Genomen
- Kompatibilitätsschwellwert δ_s einer Spezies, frei wählbar

Die Kompatibilitätsdistanz δ wird wie folgt bestimmt:

$$\delta = \frac{c_1 \cdot E}{N} + \frac{c_2 \cdot D}{N} + c_3 \cdot \overline{W}$$

E = Anzahl der excess genes
D = Anzahl der disjoint genes
$\overline{W}$ = Durchschnittliche Gewichtung der matching genes
N = Anzahl der Gene im größeren Genom
c_1, c_2, c_3 = Gewichtung, frei wählbar

III.3.2 Anpassung der Größe einer Spezies / der Population

Damit eine Populationsübernahme einer Spezies ausgeschlossen werden kann, wird ein Verfahren angewandt, explicit fitness sharing, das die Fitness einer Spezies reguliert. So wird reguliert:

1. Bestimmung der "adjusted fitness" (angepasste Fitness) für jedes Genom jeder Spezies

$$f_i^{'} = \frac{f_i}{\sum_{j=1}^{N_j} sh(\delta(i,j))}$$

$f_i^{'}$ = angepasste Fitness des Genoms i
f_i = alte Fitness des Genoms i
j = anderes Genom aus der selben Spezies wie i
$sh(\delta(i,j))$ = sharing function, gibt 0 zurück wenn $\delta(i,j) > \delta_s$, ansonsten 1

Die Funktion macht folgendes in Worten: Teile die aktuelle Fitness eines Genoms einer Spezies durch die Anzahl der Mitglieder der Spezies, die auch tatsächlich in diese Spezies gehören. Dies ist dann die angepasste Fitness des Genoms. Es können Genome in der Spezies enthalten sein bei denen auf Grund von Fitnessänderungen die Spezies, in der sie sich während der Bestimmung der angepassten Fitness befinden, nicht die Spezies ist in der sie sein sollten.

2. Anschließend wird die angepasste Fitness jedes Genoms einer Spezies aufsummiert und anschließend durch die durchschnittliche angepasste Fitness der gesamten Population geteilt. Daraus ergibt sich die Anzahl der Mitglieder in einer Spezies und somit ob die nächste Generation im Vergleich zu aktuellen wachsen oder schrumpfen soll.

$$N_j^{'} = \frac{\sum_{i=1}^{N_j} f_{ij}}{\overline{f}}$$

$N_j^{'}$ = neue Größe der Spezies
N_j = alte Größe der Spezies
f_{ij} = angepasste Fitness des Individuums i in Spezies j
$\overline{f}$ = Durchschnittliche angepasste Fitness der gesamten Population

Die fittesten Genome der Spezien werden dann gekreuzt und der Rest der Genome in der Spezies werden verworfen, sodass die Spezien ihre vorgegebene Größe $N_j^{'}$ erreichen. Die Kreuzung findet nur innerhalb einer Spezies statt. Durch dieses Verfahren wird die strukturelle Innovation geschützt, da die Populationen einer Spezies in ihrer eigenen Nische genug Zeit haben sich im Wettkampf zu optimieren.

III.4 Effizientes Suchen einer Lösung durch Minimierung der Suchräume

NEAT startet mit einer gleichförmigen Population ohne versteckte Knoten. Während die Population wächst wird, wie im vorherigen KapitelIII.3.2 beschrieben, die Größe in Spezien und somit die Größe der gesamten Population reguliert. Dafür werden die erfolgreichsten Mitglieder der Spezien gekreuzt. Die inkrementelle Entwicklung aus einer minimalen Struktur und die regelmäßige Regulierung der Größe und Mitglieder der Population sind Ursache dafür, das der Suchraum minimal gehalten wird.

IV. Anwendung

IV.1 Super Mario

In diesem Abschnitt wird erklärt, wie NEAT im Spiel Super Mario die neuronalen Netze ausbauen könnte, sodass das Spiel vom Computer gewonnen wird. Der Emulator BizHawk[6] ermöglicht es Skripte auszuführen, die in der Sprache Lua verfasst werden. Mit diesen Skripten können Emulatordaten direkt abgefragt werden (Beispielsweise Objektpositionen). Für diesen Emulator gibt es bereits ein Skript, dem NEAT als Lernverfahren zu Grunde liegt und mit dem tatsächlich das Spiel automatisch gewonnen werden kann.[7]
Wie im Kapitel "neuronale Netze" bereits beschrieben wurde können Knoten sich bei (einfachen) neuronalen Netzen in drei Ebenen

befinden. Die Bedeutung der drei Ebenen ist in Abbildung 5 erläutert.

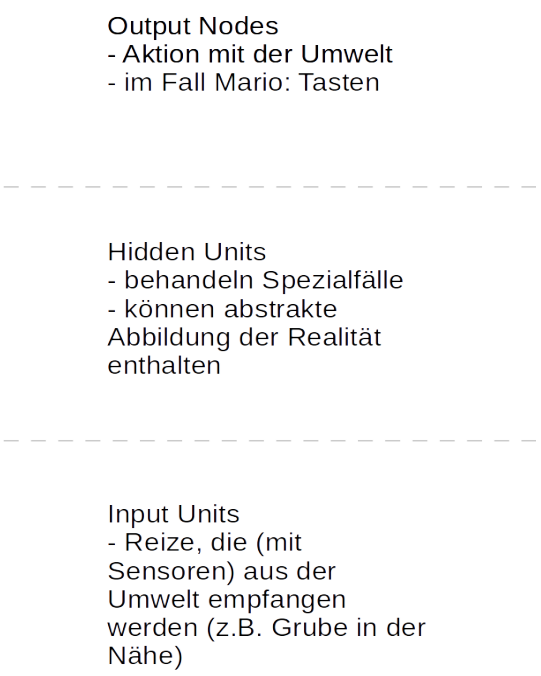

Abbildung 5: Bedeutung der Ebenen in Super Mario

Zunächst beginnt die Trainigsphase. Trainings und Testphase stehen immer in einem bestimmen Verhältnis zueinander, z.B. 80% Training und 20% Test. Im laufe dieses Kapitels wird von den Beispielwerten ausgegangen, das heißt es wird vier mal trainiert und einmal getestet, wieder vier mal trainiert und einmal getestet und so weiter. Dieser Zeitintervall (n mal Training und m mal Test) entspricht ist eine Generation (Einheit für einen Zyklus). Im ersten Schritt wird Mario versuchen aus dem Startzustand (zu Beginn des Spiels erscheinen und stehen) herauszukommen. Zu Beginn werden eine beliebige Anzahl an einfachen (minimalen) neuronalen Netzen zufällig generiert. Der richtige Weg führt sofort nach Rechts (die Taste "R" muss gedrückt werden). Diese Aktion ist in der folgenden Abbildung 6 mit einem blauen durchgängigen Pfeil gekennzeichnet (beste Lösung). Orange, gestrichelte Pfeile bedeuten, dass der Zustand sich ändert, das wünschenswerte Ergebnis der ersten Iteration jedoch nicht damit erreicht ist. Einige dieser Genome werden noch eine Weile bestehen, eventuell sogar zu guten Lösungen werden, viele werden jedoch bei der Bestimmung der Speziengrößen auf Grund von zu geringer Fitness verworfen werden. Die roten Pfeile führen nicht zu einer Zustandsänderung und das Genom wird sehr wahrscheinlich schnell entfernt werden. Der blaue Pfeil bedeutet eine maßgebliche Erhöhung der Fitness, da Mario dem Ziel einige Schritte näher kommt, bis ein Gegner, eine (gefährliche) Grube oder eine Wand erscheint.
Anmerkung: Es können auch andere Lösungen entstehen, wie Beispielsweise: "Bewege Mario nach links, neuer Zustand, es geht nämlich auf Grund einer Wand nicht weiter, bewege Mario anschließend nach rechts". Normalerweise setzt sich aber eine gute Lösung (in dem Fall sofort rechts laufen) auf Grund einer höheren Fitness im laufe der Zeit durch.

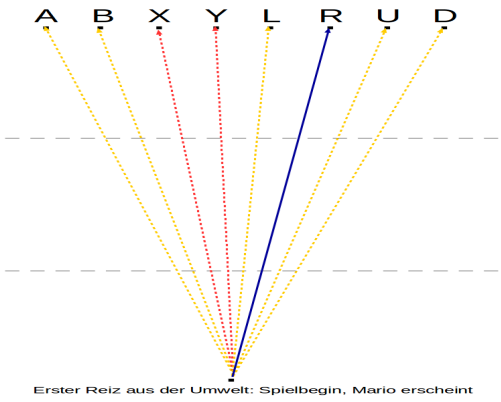

Abbildung 6: Erste Entwicklung

- A - Drehsprung
- B - [Auf Land] Springen / [Im Wasser] Hoch schwimmen
- X/Y - Beschleunigen (Mario muss sich in Bewegung befinden)

- L/R/U/D - Links / Rechts / Hoch / Runter (ducken)

IV.1.1 Trainingsphase

Ziel ist es, nach einer Aktion auf Reaktion der Umwelt zu warten (Kernidee von reinforcement learning). Die Interpretierung dieser Reaktion und die Optimierung des entstehenden Netzes macht NEAT (Ausnahme: Gewichte der Kanten werden vom separaten Algorithmus trainiert). Als Beispielnetzwerk wird das Netzwerk aus Abbildung 6 mit zwei Knoten und der blauen Kante gewählt. Die Fitness erhöht sich weiterhin bis Mario mit einer einfachen Rechtsbewegung nicht weiter kommt. Nehmen wir an Mario stößt auf eine Wand, die Fitness erhöht sich nicht mehr. Die Kollision mit der Wand erzeugt einen neuen Sensor-Knoten (Eingangsreiz aus der Umwelt). Dieser Knoten beinhaltet jetzt folgende Information: Ein Objekt wurde gesehen das nicht zu durchdringen ist. Dieser Knoten bewirkt aber noch nichts, da er mit keiner Kante verbunden ist, wodurch er andere Knoten stimulieren könnte.
Während jedem Training, genauer nach jedem Training und vor dem nächsten Training oder dem Test, werden Mutationen durchgeführt. Die Anzahl der Mutation wird vor dem ersten Training festgelegt (Beispielsweise 5% Knoten- und 10% Kantenmutationen) und wird nach jedem Training so oft für zufällige Knoten oder Kanten durchgeführt. In diesem Beispiel gibt es nun 10 Knoten (2 Sensor, 8 Output) und eine Kante. Es folgt eine "add connection mutation" für 10% der Knoten die in Frage kommen. Es können 16-1 Kanten (2*8 minus eine bestehende Kante) erstellt werden, das sind 15 Katen und 10% davon sind aufgerundet 2 Kanten die erstellt werden. Wenn man Glück hat wird der neue Sensorknoten in diesem Genom richtig verbunden (mit dem Output-Knoten der das Springen erzwingt) und zufällig auch positiv gewichtet. Sollte das in diesem Genom nicht der Fall sein, ist das nicht so schlimm, da es noch eine große Menge weiterer Genome (die sich zeitgleich entwickeln) gibt und eines dieser neuronalen Netze mit Sicherheit eine akzeptable Lösung findet. Wenn nicht, kann immer noch das kombinieren der Lösungsräume (kreuzen) gute Lösungen hervorbringen.
Es folgt noch eine "add node mutation", bei den gewählten Parametern (5% node mutations) sind es jedoch 0 Knoten die erzeugt werden (5% von 3 Kanten die aufgespalten werden können sind 0). Nur um ein Beispiel zu machen, angenommen es wären 33% so das genau ein Knoten erzeugt wird, so könnte es dazu führen das die Kante zwischen dem Sensor-Knoten "Wand" und dem Output-Knoten "Springen" in zwei Kanten aufgespalten wird (und das entsprechende Kantengen deaktiviert wird) und ein Hidden-Knoten erstellt wird. Außer das ein Knoten hinzugekommen ist ändert sich zunächst nichts, da die Kantengewichte so gesetzt wurden als ob der Knoten nie hinzugefügt wurde. Diese Gewichte können jedoch im weiteren Verlauf variieren und viel interessanter ist, dass jetzt zukünftige Kantenmutationen dazu führen können, das Spezialfälle für das Springen unterschieden werden. Zum Beispiel: Springe wenn eine Wand kommt, springe nicht sofort wenn sich unmittelbar davor eine Münze befindet.
Anschließend werden noch die Gewichte der Kanten trainiert, dies geschieht jedoch mit einem Algorithmus der von NEAT unabhängig ist (Beispielsweise lernen mittels Hebb-Regel) Das war nun eine Trainingsphase, die normalerweise einige male wiederholt wird bevor die erste Testphase kommt. Zusammengefasst ist folgendes geschehen:

- Neue Eingangsreize sind als Sensor-Knoten erschienen
- Knoten wurden durch Kantenmutationen verbunden, neue Abhängigkeiten und Verhaltensweisen entstehen
- Kanten wurden durch Knotenmutationen aufgespalten und dazwischen wurde ein neuer Hidden-Knoten hinzugefügt

- Neue Hidden-Knoten ermöglichen im weiteren Verlauf Spezialfälle zu unterscheiden
- Gewichte werden mit einem von NEAT unabhängigen Algorithmus angepasst

IV.1.2 Testphase

Als erstes wird die Fitness evaluiert. Für alle Genome wird geprüft, wie gut Sie vorgegebene Ziele erreichen oder diesen näher gekommen sind. Für Super Mario sind folgende Werte für die Fitness interessant:

- Wie nah ist Mario dem Ziel gekommen (Distanz)
- Wie schnell hat Mario die Distanz zum Ziel zurückgelegt
- Wie viele Bonuspunkte hat Mario eingesammelt

Nachdem die Fitness bestimmt wurde geht es weiter zur Spezienbildung. Zunächst werden Genome in Spezien zugeordnet. Wie im Kapitel NEAT beschrieben ist, wird für jedes Genom (A) nacheinander die Kompatibilitätsdistanz zu genau einem anderen Genom (B) jeder Spezies bestimmt, bis der Kompatibilitätsschwellwert unterschritten wird und das Genom (A) der Spezies des Genoms (B) hinzugefügt wird oder Genom (A) eine Spezies erzeugt, da keine geeignete Spezies gefunden wurde. Alle Genome werden also zunächst nach Gemeinsamkeiten sortiert. Anschließend wird für alle Genome die angepasste Fitness als aktuelle Fitness berechnet und der Durchschnitt der angepassten Fitness wird über die gesamte Population gebildet. Mit diesen Werten wird nun bestimmt welche Spezien wachsen oder schrumpfen sollen. Dabei werden Genome gekreuzt. Was Kreuzungen in Super Mario bewirken können soll ein Beispiel erläutern. Zwei der fittesten Genome (werden als Genome mit gleicher Fitness betrachtet) einer Spezies werden gekreuzt, die Ausgangssituation ist folgende:

Beide Genome haben 80% matching genes und 20% disjoint oder excess genes, die matching genes werden zufällig übernommen (wobei sich hier nur der Aktivierungszustand und das Gewicht unterscheiden können) und die 20% der disjoint und excess genes werden von beiden Genomen übernommen. Dabei kann es sein, dass eines der beiden Netze gelernt hat Hindernisse zu überspringen und das andere Netz hat gelernt Verstärkungen (z.B. Leben) einzusammeln (Mario muss dafür mit dem Kopf unter ein Objekt in der Luft springen). Das aus der Kreuzung resultierende Netz kann nun beides und da die matching genes zufällig gewählt wurden hat sich auch das generelle Verhalten geändert, da Aktivierungszustände und Gewichte der Knoten mal von einem, mal vom anderen Genom übernommen werden. Ein weiteres Training ist zur Justierung der Gewichte der Kanten erforderlich, bevor das Netz eventuell Sinnvoll agieren kann (das Netz kann auch durch die Kreuzung nutzlos geworden sein).

Das ist eine Iteration in NEAT, danach wird der Generationszähler inkrementiert und die nächste Generation wird begonnen. Im Projekt MarI/O von SethBling [7] haben die durch NEAT optimierten neuronalen Netze nur 34 Generationen gebraucht, um das Problem (Level erfolgreich beenden) zu lösen, und das sogar ziemlich gut.
Das Fazit lautet: Immer wenn ein System eine Lösung finden soll und dabei auf eine Rückmeldung der Umgebung angewiesen ist, wobei nur der Fortschritt in irgend einer Art und Weise Messbar sein muss und es egal ist, wie das Problem gelöst wird, sollte man darüber nachdenken dynamische neuronale Netze zu verwenden die durch NEAT (oder ein ähnliches Verfahren) optimiert werden.

References

[1] Rudolf Kruse, Christian Borgelt, Christian Braune, Frank Klawonn, Christian Moewes, and Matthias Steinbrecher. *Com-*

putational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Springer Vieweg, 2. auflage edition, November 2015.

[2] G.D. Rey and K.F. Wender. *Neuronale Netze: eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Aus dem Programm Huber: Psychologie-Lehrbuch. Huber, 2011.

[3] U. Rein. *Künstliche neuronale Netze und Selbstorganisation: zur Bedeutung paralleler Informationsverarbeitung für die Sozialwissenschaften*. Cuvillier, 2010.

[4] Kenneth O Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, jun 2002.

[5] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 569–577, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[6] TASVideos. Bizhawk - open source emulator for video gaming consoles. `https://github.com/TASVideos/BizHawk`, 2014.

[7] SethBling. Mari/o - machine learning for video games. `https://www.youtube.com/watch?v=qv6UVOQ0F44`, 2015.