fhwedel
UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE WEDEL

SEMINARARBEIT

in der Fachrichtung

Medieninformatik

Thema:

Spielstrategien

Eingereicht von:	Nils Böckmann Schinkelring 110 22844 Norderstedt Tel. (040) 526 17 44
Erarbeitet im:	6. Semester
Abgegeben am:	22.06.2005
Referent:	Prof. Dr. Sebastian Iwanowski Fachhochschule Wedel Feldstrasse 143 22880 Wedel Tel. (04103) 804 8 – 63

1 Einführung

1.1 Problemstellung

Spiele faszinieren die Menschheit schon seit den frühen Anfängen. Sie bringen Entspannung, Freude und Ablenkung vom Alltag. In der heutigen Gesellschaft ist es jedoch immer seltener möglich, dass sich Menschen zum Spielen treffen. Der Mensch möchte spielen, wenn er dazu Zeit und Lust hat. Aus diesem Grund werden künstliche Computerspieler immer stärker gefördert und nehmen immer größere Teile der Forschung in Anspruch. Eine Schwierigkeit dieses Ansatzes ist jedoch, dass der Mensch beim Spielen gefordert werden möchte. Die Computerspieler müssen daher in der Lage sein, den Menschen zu schlagen. Techniken wie sie dies erreichen können, sind Gegenstand dieser Ausarbeitung.

1.2 Abgrenzung

In dieser Arbeit werden die drei wesentlichen Konzepte zur Realisierung eines künstlichen Computergegners vorgestellt. Diese drei Ansätze lassen sich grob in zwei Kategorien unterteilen. Es gibt Verfahren, die mit einem deklarativen- und solche die mit einem heuristischen Mechanismus arbeiten. Die heuristischen Verfahren werden in dieser Arbeit sehr genau diskutiert, die deklarativen nur kurz vorgestellt.

1.3 Zielsetzung

Nach dem Lesen dieser Ausarbeitung soll es dem Leser möglich sein, einen einfachen Computergegner mit der Technik des Spielbaumes zu implementieren. Weiterhin soll ein Überblick über die verschiedenen zur Verfügung stehenden Techniken und deren Einsatzgebiete entstehen, so dass die Vor- und Nachteile transparent werden.

2 Was ist eine Spielstrategie

„Eine Spielstrategie bezeichnet ein Verfahren, mit dem es dem Computer möglich ist, komplexe Spielregeln in Form von Regeln oder heuristischen Funktionen abzubilden und damit eine Bewertungsgrundlage zu schaffen an Hand derer über das nächste Handeln entschieden werden kann.“

3 Die drei Verfahren

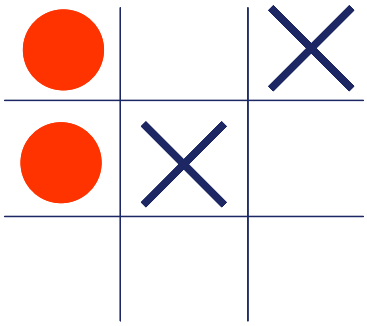
3.1 Umgebungslogik

3.1.1 Grundlegende Informationen

Die Umgebungslogik kann in zwei Bereiche unterteilt werden. Es gibt den deklarativen und den lernenden Ansatz. In beiden Ansätzen werden Wissensbasen über das Spiel formuliert. Diese Wissensbasen bestehen aus Regeln, die die Regeln des zu implementierenden Spiels widerspiegeln. Diese Regeln nennt man Sätze. Jeder dieser Sätze besitzt Handlungsanweisungen, die für eine bestimmte Situation des Spiels geeignet sind.

3.1.2 Beispiel „Tic Tac Toe“

Betrachten wir das kleine Spiel „Tic Tac Toe“. Klein kann man das Spiel nennen, weil es nur aus neun Feldern besteht und sehr einfache Regeln besitzt. In der folgenden Situation gibt es für den Computer (rote Kreise) nur eine sinnvolle Möglichkeit für den nächsten Zug. Er sollte nach unten Links ziehen. Die Aussagenlogische Formel hierfür lautet „*A1 und A4 -> A7*“



A1	A2	A3
A4	A5	A6
A7	A8	A9

Nun muss noch dafür Sorge getragen werden, dass der Computer auch die anderen sieben Situationen erkennt, die diesem Muster folgen. Es werden also 8 Muster benötigt, nur um alle Fälle abzudecken in denen zwei Steine direkt nebeneinander liegen und der dritte zum Sieg gesetzt werden kann.
Es müssen jedoch noch die zwei anderen Fälle, „Rot-Frei-Rot“ und „Frei-Rot-Rot“ die auftreten können berücksichtigt werden. Daher werden alleine für jede Gewinnsituation die es für den Computer geben kann 24 Regeln benötigt.

3.1.3 Bewertung

Obwohl es bei „Tic Tac Toe“ nur sehr wenige Regeln und Felder gibt, entstehen schon sehr viele Regeln. Würde man versuchen einen Schachcomputer mit Hilfe einer Umgebungslogik zu implementieren, würde die Anzahl der Regeln explodieren. Daher ist dieser Ansatz in der Praxis nicht verwendbar, da es erforderlich ist, jede mögliche Spielsituation eines Spieles zu generieren und für diese eine Handlung vorzusehen.

3.2 Spielbäume

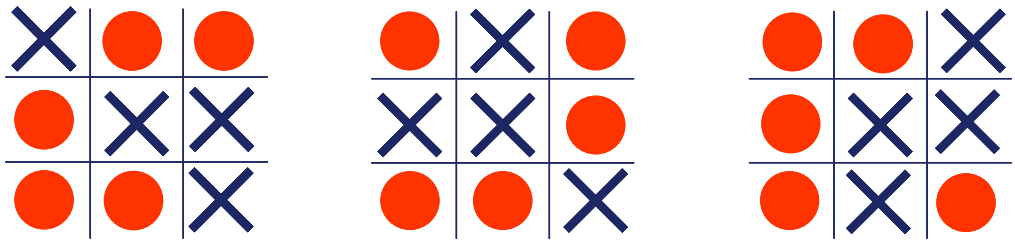
3.2.1 Grundlegende Informationen

Spielbäume errechnen ausgehend von der aktuellen Spielposition alle möglichen folgenden Positionen und bewerten diese dann mit Hilfe einer Funktion (der Bewertungsfunktion). Hierzu verwenden Sie die enorme Rechenleistung der heutigen Computer. Nach dem generieren des Spielbaumes wird in ihm, mit Hilfe des „Min/Max Verfahrens“, nach dem bestmöglichen Zug für den Computer gesucht. Das angestrebte Ziel ist, den Spielverlauf so weit wie möglich vorauszuberechnen. Bei kleinen Spielen wie „Tic Tac Toe“ ist dies bis zum Endstand möglich, da nur 8! (40320) Knoten nach dem ersten Zug zu berechnen sind. Bei Spielen wie Schach oder Dame ist dies nicht mehr möglich. Bei diesen Gesellschaftsspielen muss das expandieren des Baums ab einer gewissen Stufe abgebrochen werden. Durch dieses Vorgehen entsteht das Horizontproblem. Wenn die Suche an einem gewissen Spielstand abgebrochen wird, kann der Computer genau wie ein Mensch wichtige Dinge übersehen, die sich erst im weiteren Spielverlauf herausstellen würden.

3.2.2 Die Bewertungsfunktion

Mit Hilfe der Bewertungsfunktion schätzt der Computer die Güte der Lage der aktuell errechneten Spielposition ein. Sie ist der zentrale Aspekt einer Spiellogik. Wenn Sie kein Wissen über das Spiel enthält, dann kann auch die Spiellogik zu keinem Erfolg führen. Auch hier lassen sich wieder die Beispiele „Tic Tac Toe“ und Schach anführen. Bei „Tic Tac Toe“ ist es sehr einfach eine Spielposition zu bewerten, bei Schach ist das nicht mehr so trivial. Dort fließen sehr viele Faktoren in die Einschätzung der momentanen Lage ein. Zur Verdeutlichung der Arbeitsweise nehmen wir als Beispiel die drei Endsituationen des Spiels „Tic Tac Toe“. Als „Tic Tac Toe“ spielender Mensch lässt sich leicht erkennen wer gewonnen, verloren oder ein Unentschieden errungen hat. Dem Computer muss diese Fähigkeit antrainiert werden. In dem Programm legen wir fest, dass drei rote Kreise in einer Reihe gut sind. Daher bekommt dieser Fall eine hohe Bewertung innerhalb unseres Systems. Weiterhin legen wir fest, dass drei Kreuze in einer Reihe schlecht für uns sind. Dieser Fall erhält daher eine sehr schlechte Bewertung von uns. Der dritte Fall der existieren kann ist ein

Unentschieden. Wir spielen lieber unentschieden als zu verlieren, jedoch gewinnen wir lieber als unentschieden zu spielen. Daher erhält diese Spielsituation eine Bewertung in der Mitte. Da wir in diesem Spiel nur drei verschiedene Endzustände haben, können wir die Niederlage auf minus eins, das Unentschieden auf Null und den Sieg auf eins definieren. In komplexeren Spielen muss diese Wertung nicht so einfach gehalten sein. Dort werden sehr häufig auch Gleitkommazahlen zur besseren Differenzierung verwandt.

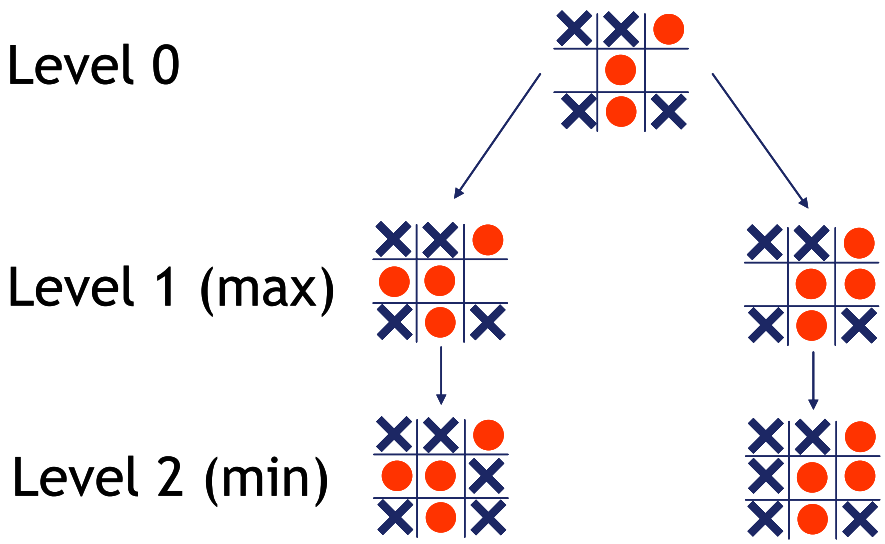


(-1) (0) (1)

3.2.3 Das Min/Max Verfahren

Mit der Bewertungsfunktion haben wir nun die Grundlage geschaffen, um den nächstbesten Zug für uns zu bestimmen. Es ist jedoch noch nicht möglich, das Zug/Gegenzugverhalten während eines Spiels abzubilden. Hierzu wird das Min/Max Verfahren verwendet. Mit Hilfe von ihm ist es möglich, die verschiedenen Spielzüge der unterschiedlichen Spielparteien abzubilden. Der Name dieses Verfahrens stammt daher, dass die Bewertungsfunktion die Spielstände stets aus sicht des Computers beurteilt.

Das Verfahren beginnt mit der untersten Knotenschicht im Baum. Wenn eine gerade Schicht vorliegt, dann ist dies eine MIN-Schicht, wenn eine ungerade vorliegt, eine MAX-Schicht. Innerhalb einer Maximalen Schicht, reichen wir das Ergebnis weiter nach oben, welches die höchste Wertigkeit



besitzt. Diese Vorschrift repräsentiert den eigenen Zug des Computers, denn dieser möchte stets den für sich besten Zug machen. Bei der minimalen Schicht ist dies umgegehrt analog. Dies repräsentiert den Zug des Gegenspielers. Dort wird auch davon ausgegangen, dass er den für sich bestmöglichen Zug macht. Da die Bewertung stets aus Sicht des Computers erfolgt, wählt er also den für uns schlechtmöglichsten Zug.
In diesem Beispiel würde der Algorithmus erst zur linken MIN-Schicht absteigen und dort eine Bewertung mit der Wertigkeit null finden. Diese würde zum nächsten Knoten hoch gereicht werden. Da dann alle Knoten dieser Seite expandiert wurden, wird nun der rechte Teil des Baumes untersucht. Auch hier wird mit der untersten Schicht begonnen. Dort wird eine Bewertung mit der Wertigkeit minus eins gefunden. Diese wird nach oben gereicht. Da dann alle Knoten im Baum abgesucht wurden, wird nun auf der ersten Schicht der maximale Wert gesucht. Der Knoten, der durch diesen Wert repräsentiert wird, ist die beste Wahl, da das Ziel des Algorithmus in einer MAX-Schicht die Maximierung der Wertigkeit innerhalb eines Knotens ist.

3.2.4 Ansätze zur Laufzeitverkürzung

3.2.4.1 Alpha/Beta Pruning

Die Technik des „Alpha/Beta-Abschneidens“ macht sich eine Eigenschaft der Definition des „Min/Max Verfahrens“ zu Nutze. Es werden zwei Grenzen eingefügt, die die momentan garantierte Punktzahl (Alpha Grenze) und die momentan minimal zugelassene Punktzahl (Beta Grenze) abbilden.

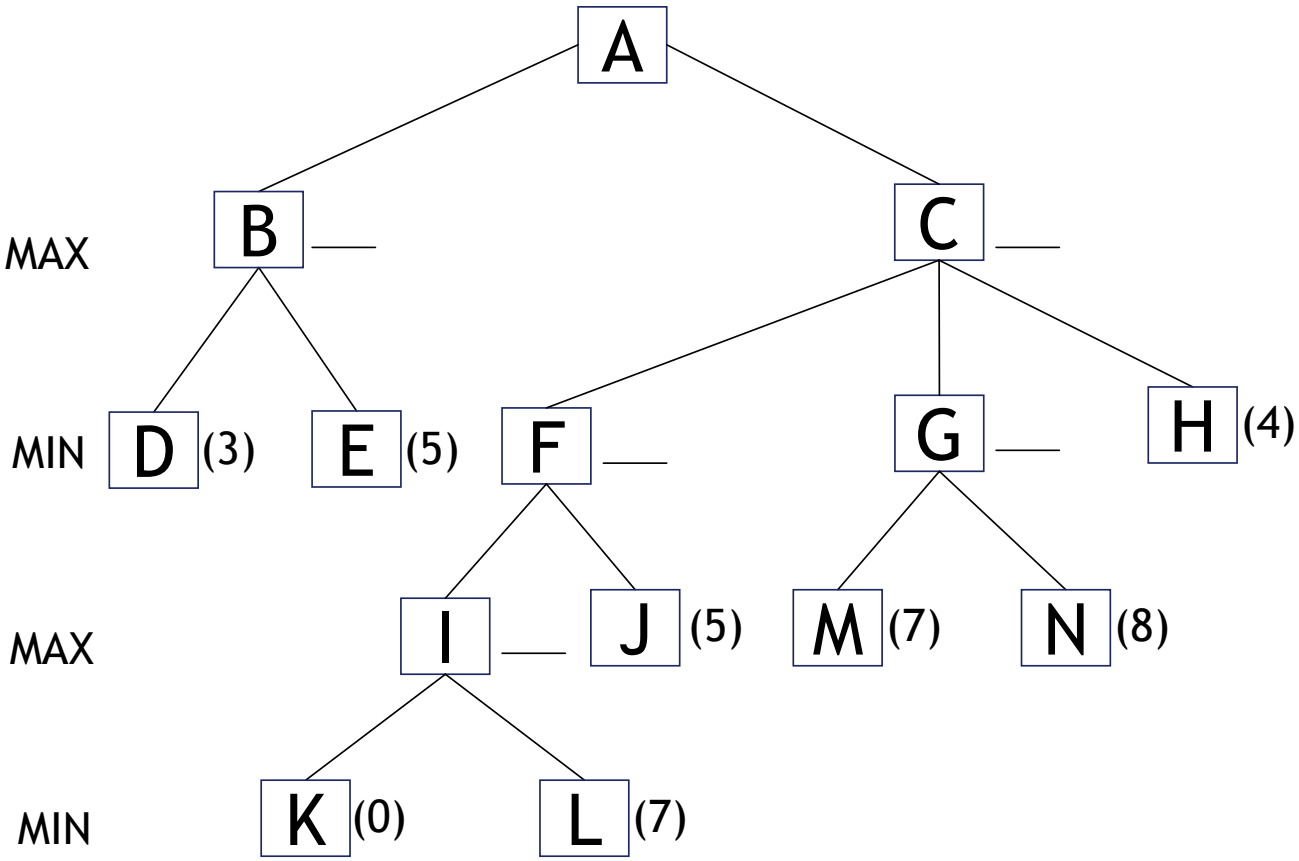
3.2.4.1.1 Alpha Grenze

Wenn in einer minimalen Schicht ein Wert gefunden wird, der geringer ist als ein Wert, der zuvor in einer minimalen Schicht gefunden wurde, gibt es einen Zug, aus dem eine bessere Punkteausbeute folgt, als aus dem aktuell untersuchten. Wenn dies der Fall ist, ist es nicht mehr nötig die weiteren Kinder des Vaterknotens des aktuellen Knotens zu untersuchen, da das Ergebnis nur noch schlechter werden kann (*Alpha Abbruch*). Wenn in einer minimalen Schicht ein Wert gefunden wird der größer als die aktuelle Alpha Grenze ist, wird die Grenze aktualisiert und der Wert wie beim Min/Max Verfahren nach oben gereicht.

3.2.4.1.2 Beta Grenze

Wenn in einer maximalen Schicht ein Wert gefunden wird, der größer ist als ein zuvor gefundener Wert einer maximalen Schicht, kann davon ausgegangen werden, dass der Gegenspieler diesen Zug nicht machen wird, da er für ihn nicht optimal ist. Aus diesem Grund wird die Beta Grenze eingeführt. Gegen diese Grenze wird jeder Wert einer maximalen Schicht verglichen. Sollte der Wert größer sein als die aktuelle Beta Grenze, wird die Untersuchung des Vaterknotens des aktuellen Knotens und aller Nachfolger des Vaterknotens übersprungen (*Beta Abbruch*). Sollte der Wert kleiner als die Beta Grenze sein, wird die Beta Grenze aktualisiert und der Wert nach oben gereicht.

3.2.4.1.3 Beispiel „Tic Tac Toe“[1]



In diesem Beispiel soll der oben gezeigte Baum mit Hilfe des Alpha/Beta Pruning Verfahrens verarbeitet werden. Der Algorithmus nimmt sich stets den linken Teilbaum zuerst vor. Am Anfang des Verfahrens ist lediglich Knoten A bekannt, die anderen Knoten werden erst später expandiert. Dies wird in der grafischen Darstellung zu diesem Beispiel vernachlässigt.

Als erstes wird Knoten B untersucht. Ausgehend von diesem geht der Algorithmus zum Knoten D. Der Spielstand in diesem Knoten wird von unserer Bewertungsfunktion mit dem Wert drei beziffert. Da noch kein Wert in Knoten B steht, kann dieser Wert sofort übernommen werden. Des Weiteren wird die Alpha Grenze aktualisiert, da auch diese noch leer ist. Nun wird Knoten E expandiert. In ihm wird die Fünf gefunden. Da der Knoten sich in einer minimalen Schicht befinden und bereits eine drei gefunden wurde, wird dieser Wert nicht weiter beachtet. Der komplette linke Teilbaum ist damit abgearbeitet und die Bewertung für den Spielzug B steht fest. Nun wird der rechte Teilbaum untersucht. Dort werden alle Knoten expandiert, bis der

[1] KI-Einführung und Anwendungen (Elaine Rich)

Knoten K gefunden wird. Dieser Knoten wird mit einer null bewertet. Da der Knoten sich in einer minimalen Schicht befindet, wird sein Wert gegen den der Alpha Grenze verglichen. Da der Wert des Knoten K geringer ist, als der der Alpha Grenze, erfüllt er das Kriterium eines Alpha Abbruchs. Daher wird der Kindknoten L nicht mehr untersucht. Der nächste Knoten der Agenda ist Knoten J. In ihm wird eine fünf von der Bewertungsfunktion gefunden. Da der Knoten sich in einer maximalen Schicht befindet und die Beta Grenze noch keinen Wert hat, wird diese mit dem Wert fünf initialisiert. Außerdem wird der Wert fünf an den Knoten F hoch gereicht, der diesen dann an den Knoten C weitergibt. Ausgehend vom Knoten C wird nun in Richtung des Knotens M expandiert. Da dieser Knoten in einer maximalen Schicht liegt, wird sein Wert gegen den der Beta Grenze verglichen. Da der Wert höher ist als der Wert der Beta Grenze erfüllt dieser Knoten das Kriterium eines Beta Abbruchs. Aus diesem Grund kann die Auswertung von dem Knoten N übersprungen werden. Der Wert des Knotens C muss nicht aktualisiert werden. Auf unserer Agenda ist nun lediglich Knoten H verblieben, zu dem nun expandiert wird. Die Spielsituation des Knotens wird von der Bewertungsfunktion mit dem Wert vier beziffert. Da sich der Knoten innerhalb einer minimalen Schicht befindet, wird sein Wert gegen die Alpha Grenze verglichen. Dieser Vergleich ergibt, dass drei nicht die maximale Punktzahl ist, die wir dem Gegner „wegnehmen“ können. Daher werden die Alpha Grenze, sowie der Wert des Knotens C aktualisiert. Danach steht die Wahl des besten Zuges fest. Der Computer wird zum Knoten C ziehen.

3.2.4.2 Weitere Optimierungen

Eine Möglichkeit zur Optimierung der Laufzeit von Spielbäumen ist die Einschränkung des Suchverfahrens auf eine bestimmte Anzahl von Ebenen innerhalb des Spielbaums. Dies ist jedoch mit Konsequenzen verbunden, die mit weiteren Optimierungen wie der bestätigenden Zugauswahl wieder abgeschwächt werden sollten. Mit Hilfe dieses Verfahrens ist es möglich das Horizontproblem weitestgehend zu eliminieren. Bei der bestätigenden Zugauswahl sucht der Algorithmus innerhalb bestimmter Grenzen den Baum nach dem besten Spielzug ab. Wenn dieser gefunden wurde, wird der gewählte Zug durch eine erneute Suche ab diesem Knoten bestätigt oder verworfen, hierdurch lässt sich sehr viel Zeit und Speicher sparen.

Bei Spielen wie Schach ist die Anwendung von gespeicherten Eröffnungs- und Schlusszügen sehr verbreitet. Dies lässt sich gut Realisieren, da sich gerade die Anfangssituationen von Schach stark ähneln.

3.2.5 Bewertung

Spielbäume sind ein mächtiges Werkzeug um eine künstliche Intelligenz für Spiele zu implementieren. Ein großer Vorteil dieser Technik ist die Wiederverwendbarkeit der Algorithmen. Sind die Algorithmen für das „Alpha/Beta Pruning“ bzw. das „Min/Max Verfahren“ und die Baumstruktur einmal implementiert, müssen für jedes Spiel nur noch die Bewertungs- und Spielzuggenerierungsfunktionen neu implementiert werden.
Eine Schwierigkeit der Spielbäume ist sicherlich die Bewertungsfunktion. Es ist nicht immer trivial für ein Spiel eine Rechenvorschrift zu erstellen, die jede Spielsituation korrekt Einschätzen kann.

3.3 Künstliche Neuronale Netze

3.3.1 Grundsätzliche Informationen

Künstliche Neuronale Netze versuchen das Gehirn des Menschen nachzubilden. Sie werden häufig in der Bildverarbeitung, Schrifterkennung, Spracherkennung, Regelungstechnik und bei Spielen genutzt. Ein künstliches neuronales Netz besteht aus einer Eingabe- und Ausgabeschicht. Optional können zwischen diesen beiden Schichten noch versteckte Schichten (hidden layers) liegen. Ausgehend von der Eingabeschicht sind in einem „Feed Forward Netzwerk“ alle Knoten einer Schicht mit allen Knoten der nächsten Schicht verknüpft. Jeder Knoten enthält daher mehrere Eingänge und gibt seinen Ausgabewert selber wieder an mehrere Knoten ab. Wichtig ist hierbei, dass sich die Eingabewerte unterscheiden können, der Ausgabewert aber stets der Selbe ist. Damit ein Neuron feuert, benötigt es eine gewisse Aktivierungsenergie. Diese Aktivierungsenergie wird für jeden Knoten einzeln eingestellt. Durch das Verändern dieser Gewichte wird das Netzwerk an die Funktion angepasst, die es darstellen soll.

3.3.1.1 Arten von neuronalen Netzen[2]

- **Perceptron**
    - Die Eingabe und Ausgabeschicht sind direkt verbunden
- **Einschichtige neuronale Netze:**
    - Neuronen liegen direkt zwischen Eingabe und Ausgabe
- **Mehrschichtige neuronale Netze:**
    - Es gibt Zwischenschichten, die “im Verborgenen” sind
- **Neuronale Netze mit Rückkopplung:**
    - Ausbildung eines “Gedächtnisses”

[2] http://www.fh-wedel.de/~iw/Lehrveranstaltungen/WS2004/WBS/WBS12.pdf

3.3.1.2 Das Neuron

Das Neuron besteht aus verschiedenen Eingabewerten, einem Gewicht und gleichen Ausgabewerten. Um zu Überprüfen, ob das Aktivierungslevel des Knotens erreicht ist, werden alle Eingabewerte summiert, diese Summe dann mit dem Gewicht verrechnet und der Wert der sich ergibt in die Aktivierungsfunktion eingegeben. Diese wertet die Eingabe entsprechend aus und gibt, sollte der Aktivierungslevel erreicht worden sein, einen Impuls über seinen Ausgabeverknüpfungen an die nächste Schicht weiter.

3.3.1.3 Arten von Aktivierungsfunktionen

Es gibt vier Arten von Aktivierungsfunktionen, die sehr gebräuchlich sind. Das ist zum einen die „Step Funktion“, die ab einem gewissen Wert eins liefert und davor den Wert null ausgibt. Des Weiteren existiert die „Sign Funktion“, die je nach Eingabewert eine minus eins oder eine eins liefert. Am Ende bleiben noch die stetigen Funktionen über. In dieser Kategorie gibt es die sigmoiden und die hyperbolischen Funktionen.

3.3.2 Mustergenerierung

3.3.2.1 Manuell

Bei der manuellen Mustererzeugung werden zu einem gegebenen Eingabevektor die erwarteten Ausgabevektoren gespeichert. Damit hat man eine „Rechenvorschrift“, die festlegt, welche Ausgabewerte zu gewissen Eingabewerten erwartet werden. Diese Muster sind bei Spielen wie „Tic Tac Toe“ sehr einfach zu erzeugen, der Programmierer baut während der Erzeugungsphase gewisse Spielsituationen auf und legt dann die Reaktion auf diese Situation fest. Mit Hilfe dieser Muster wird das Netz später trainiert.

3.3.2.2 Teilautomatisch

Die teilautomatische Mustergenerierung funktioniert grundsätzlich wie die automatische Generierung von Mustern. Der Unterschied zu der automatischen Generierung besteht darin, dass ein Mensch gegen das künstliche neuronale Netz spielt und dass die Muster innerhalb der Partie gelernt werden. Das Netz

lernt somit während des Spielens. Alle weiteren Prinzipien sind im Abschnitt automatische Mustererzeugung zu lesen.

3.3.2.3 Automatisch

Bei der automatischen Mustererzeugung werden die Muster nicht von einem Programmierer erzeugt, sondern entstehen dadurch, dass das künstliche neuronale Netz gegen sich selber antritt. Die Muster, die im Laufe einer Partie entstehen werden gespeichert und nach Beendigung des Spiels werden diese dann abhängig von dem Ausgang des Spiels bewertet. Diese Bewertung kann entweder bestärkend (das zu trainierende Netz hat gewonnen), neutral (unentschieden), oder negativ verstärkend sein (das zu trainierende Netz hat verloren). Die letzen beiden Fälle sind interessant. Sollte es dem Netz nicht gelungen sein zu gewinnen, ist es nötig, dass die Gewichte neu eingestellt werden. Um dies auf das Prinzip der Muster abbilden zu können, werden die entsprechenden Ausgabevektoren angepasst, so dass sie beim späteren Training wieder nach dem Prinzip des Teached Training[3] verwendet werden können.

3.3.3 Training von künstlichen neuronalen Netzen

Beim Training von künstlichen neuronalen Netzen gibt es drei wichtige Faktoren.

- **Trainingsdurchläufe**
    - Die Durchläufe, die während einer Trainingsphase durchgeführt werden
- **Threashold (Trainingsgrenze)**
    - Der Fehler, bei dem das Trainieren abgebrochen wird
- **Teaching Factor (Intensität des Lernens)**
    - Gibt an, wie stark der Fehler ins Netz zurück propagiert werden soll. Wird dieser Wert zu gering oder hoch gewählt, lernt das Netz nicht optimal.

[3] Siehe 3.3.3.1.1

3.3.3.1 Trainingsverfahren

3.3.3.1.1 Teached Training

Beim Teached Training wird das Netz mit den zuvor erzeugten Mustern trainiert. Das Verfahren läuft in einer Schleife die dem Netz immer wieder alle Muster präsentiert und die Ausgaben des Netzes mit denen der Muster vergleicht. Das Ziel bei diesem Verfahren ist die Minimierung des Fehlers. Sollte das Netz bei einem Inputvektor eines Musters einen Outputvektor der ungleich des Erwarteten ist erzeugen, werden die Gewichte der beteiligten Knoten mit Hilfe des Backpropagation Algorithmus neu eingestellt. Dies wird so lange wiederholt, bis das Netz den gewünschten minimalen Fehler (Threshold) oder die maximale Anzahl der Iterationen erreicht hat.

3.3.3.1.2 Trainieren mit „Mutierten" Netzen

Das trainieren mit „mutierten" Netzen ist eine Form des bestärkenden Lernens. Damit diese Art des Trainings funktionieren kann ist es am besten, wenn das zu Trainierende künstliche neuronale Netz schon mit Hilfe des teached Training antrainiert worden ist. Es werden zwei neuronale Netze benötigt. Das erste neuronale Netz ist das unveränderte Netz, welches trainiert werden soll. Das zweite Netz ist ein Netz, das aus dem ursprünglichen Netz mit Hilfe von einer Funktion (häufig dem Gauß) abgeleitet worden ist. Es ist daher anders als das ursprüngliche Netz, jedoch noch mit ihm verwandt. Dieses „mutierte" Netz, in dem die Gewichte der einzelnen Knoten verändert wurden, tritt nun gegen das ursprüngliche Netz an. Sollte sich nach einigen tausend Spielen die Tendenz zeigen, dass das „mutierte" Netz besser als das alte ist, wird die neue Generation des Netzes übernommen und der Vorgang wiederholt. Bei „Tic Tac Toe" macht es Sinn den Vorgang so lange zu wiederholen, bis kein Netz mehr gewinnt, da es bei „Tic Tac Toe" keinen Sieger geben kann wenn beide Parteien optimal spielen. Diese Überprüfung sollte dann aber zwischen den gleichen Netzen stattfinden, damit nicht die Willkürlichkeit des anderen verwendeten Netzes das Ergebnis verfälscht.

3.3.4 Beispiel „Tic Tac Toe“[4]

Eine Architektur für ein künstliches neuronales Netz könnte wie folgt aufgebaut sein.

- 12 Neuronen in der Eingabeschicht
- 12 Neuronen im 1. Hidden Layer
- 06 Neuronen im 2. Hidden Layer
- 01 Neuron in der Ausgabeschicht

Eine Erklärung für diese Werte lässt sich nicht finden. Sie sind durch Ausprobieren entstanden. Eine weitere wichtige Entscheidung bei der Entwicklung von künstlichen neuronalen Netzen ist die Wahl der Eingabewerte. In diesem Beispiel werden zwölf Eingabeneuronen verwendet. Die Belegung sieht wie folgt aus:

- Player 1 Einer
- Player 1 Zweier
- Player 1 Dreier
- Player 2 Einer
- Player 2 Zweier
- Leer -
- Mixed Zweier / 10
- Mixed Dreier / 10
- Free Lines / 10
- Free Corners / 2 -1
- Free Center *2 -1
- Free Fields / 10

Diese Größen sind leichter zu finden als die Dimensionen des Netzes, denn über diese Werte wird versucht dem neuronalen Netz genügend Informationen über die aktuelle Spielsituation zu liefern. Es müssen daher Kriterien ausgewählt werden, die Aussagekraft über ein Spiel haben.
Als Aktivierungsfunktion wurde in diesem Beispiel eine hyperbolische Funktion gewählt (f(x)=tanh(x))

3.3.5 Bewertung

Wenn ein künstliches neuronales Netz erst einmal funktioniert, dann liefert es nahezu so gute Ergebnisse wie Spielbäume. Doch bis zu diesem Zustand ist es ein langer Weg. Es müssen die optimale Anzahl von Knoten, korrekte Aktivierungsfunktionen gefunden und die Gewichte dem speziellen Problem angepasst werden. Der schwierigste Punkt dieser langen Liste ist die Wahl der

[4] http://www.informatik.htw-dresden.de/~iwe/Belege/2004/Kuehne

korrekten Anzahl von Neuronen. Wenn zu viele Knoten verwendet werden, generalisiert das Netz nicht, sondern lernt die antrainierten Fälle auswendig, schafft es aber nicht sein Wissen auf neue Fälle zu übertragen. Sollten zu wenige Knoten gewählt werden, ist das Netz nicht im Stande zu lernen. Für dieses Problem gibt es jedoch ein Verfahren, dass versucht die Optimale Knotenanzahl im Netz zu herzustellen. Dieses Verfahren nennt sich „Optimal Brain Damage“. Wenn dieses Verfahren verwendet werden soll, sollte das Netz bewusst mit zu vielen Knoten gestartet Werden. Der „Optimal Brain Damage“ Algorithmus versucht nun die Knoten innerhalb des Netzes zu bestimmen, die nicht zur Erzeugung eines Ergebnisses beitragen (Gewichte nahe oder gleich null). Diese Knoten werden aus dem Netz gelöscht. Dann wird das neu entstandene Netz gegen das Netz getestet, welches noch alle Knoten enthält. Sollten die Ergebnisse gleich gut oder besser sein, wird das neue Netz übernommen und der Vorgang kann wiederholt werden. Mit diesem Verfahren ist es möglich bis zu 1/3 der Anfangsknoten einzusparen.

4 Fazit

Es gibt viele verschiedene Techniken, die alle zu verschiedenen Ergebnissen führen. Festzustellen bleibt, dass mit der Spielbaumtechnik und den künstlichen neuronalen Netzen wohl die besseren Ergebnisse erzielt werden können, als mit Umgebungslogiken. Wobei mir persönlich der Ansatz der Spielbäume besser gefällt, da hier ein überschaubarer, funktionaler Ansatz verwendet wird. Andere mögen dies als Nachteil sehen, da es häufig einfacher ist einen gewünschten Output mit Hilfe von Mustern zu trainieren. Ich jedoch halte die deterministische Vorgehensweise für die meisten Probleme besser geeigneter.
Ein echter Vorteil der künstlichen neuronalen Netze ist die kürzere Laufzeit im Vergleich zu Spielbäumen. Des Weiteren lassen sich auch Probleme lösen, für die es keinen Funktionalen Ansatz gibt.
Bleibt am Ende zu sagen, dass die Implementierung einer künstlichen Intelligenz viel mit Philosophie zu tun hat und jeder seine eigenen Vorlieben hat. Ganz wichtig ist: „die künstliche Intelligenz ist immer nur so gut, wie sein Programmierer das Spiel verstanden hat“.

5 Literaturverzeichnis

Bücher

[1] Artificial Intelligence – A modern Approach (Stuart Russel / Peter Norvig)

[2] Strategische Computerspiele für Ihren ATARI (John White)

[3] KI-Einführung und Anwendungen (Elaine Rich)

[4] Essentials Of Artificial Intelligence (Matt Ginsberg)

Internet

[5] http://www.fh-wedel.de/~iw/Lehrveranstaltungen/WS2004/WBS/WBS12.pdf

[6] http://www.grundstudium.info/neuro/

[7] http://www.wikipedia.de

[8] http://www.informatik.htw-dresden.de/~iwe/Belege/2004/Kuehne

[9] http://www.mathematik.de/spudema/spudema_beitraege/beitraege/vorberger/

[10] http://student.cosy.sbg.ac.at/~amayer/projects/betagammon/backgammon.pdf

[11] http://www.research.ibm.com/deepblue/