Training Künstlicher Intelligenzen im *Vier Gewinnt*-Spielen

Sebastian Dorn

15. Juli 2012

Zusammenfassung

Es werden verschiedene Künstliche Intelligenzen (KI) mit überwachten Lernverfahren darin trainiert, das Brettspiel *Vier Gewinnt* zu spielen. Getestet werden das Multilayer Perceptron (MLP), das Radial Basis Function Network sowie der Decision Tree. Die Leistungen der verschiedenen KIs im Spiel werden evaluiert, wobei sie auch mit größeren und kleineren Spielbrettern konfrontiert werden, als in den Trainingsdaten vorgesehen. Im Vergleich zeigt sich dann, dass von den trainierten KIs nur das MLP einen guten Gegenspieler liefert.

1 Einleitung

Vier Gewinnt ist ein für zwei Spieler ausgelegtes Brettspiel und findet auf einem Spielfeld von 7x6 Feldern statt, wie es in Abbildung 1 zu sehen ist. Abwechselnd wählen die Spieler eine Spalte, in die sie einen Spielstein ihrer Farbe werfen. Spielsteine fallen in der Spalte hinunter bis zum letzten freien Feld. Es gewinnt der Spieler, der zuerst eine Kette aus vier seiner Steine gebildet hat. Dabei wird entweder horizontal, vertikal oder diagonal gezählt.

Für die standardmäßige Spielfeldgröße wurde das Spiel bereits gelöst. Das heißt, es gibt perfekte Spieltaktiken und somit lässt sich – vorausgesetzt jeder Spieler folgt der jeweils bestmöglichen Taktik – das Ergebnis einer Partie anhand des Eröffnungssteines vorhersagen [3].

Ziel dieser Arbeit ist es, verschiedene Künstliche Intelligenzen (KI) im Spiel *Vier Gewinnt* zu trainieren und ihre erlernten Fähigkeiten im Spiel gegen menschliche Spieler zu testen.

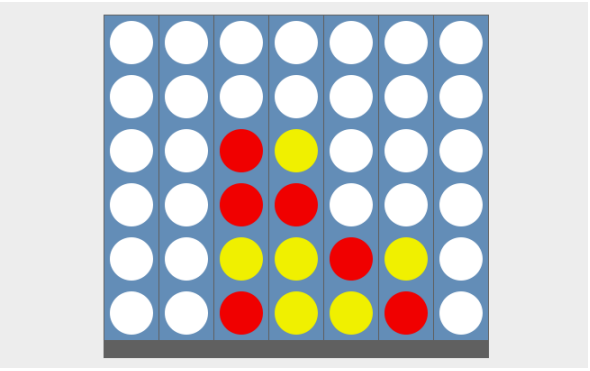

Abbildung 1.1: Vier Gewinnt. Spieler Rot hat gewonnen, da er als Erster vier seiner Steine gemäß den Regeln verbinden konnte.

Anschließend wird betrachtet, wie die KIs mit Spielfeldern zurechtkommen, die in ihrer Größe nicht dem Standard entsprechen. Sie werden somit mit einer Situation konfrontiert, für die sie nicht speziell trainiert wurden, die aber dennoch im Rahmen des Problem-Szenarios (Spielen von *Vier Gewinnt*) liegt.

2 Verwandte Arbeiten

In einem Vergleich eines Multilayer Perceptrons (MLP) und eines K-Nearest-Neighbour-Algorithmus im Rahmen von Bestärkendem Lernen (engl. *Reinforcement Learning*) wurde festgestellt, dass das MLP *Vier Gewinnt* effizienter lernte und im Spiel gegen Menschen sehr gute Ergebnisse lieferte [2].

Eine Arbeit, die sich nur darauf konzentrierte, einem MLP *Vier Gewinnt* beizubringen, kam ebenfalls zu einem positiven Ergebnis. Dabei wurden verschiedene Konfigurationen getestet und unter anderem beobachtet, dass Training mit zu vielen – zuvor aufgenommenen – Spielpartien den Lernprozess mitunter stark minderte. Des Weiteren passten sich Netzwerke mit wenigen versteckten Neuronen (84 Neuronen) zu langsam an, während Netzwerke mit vielen versteckten Neuronen (672 Neuronen) zu starken Schwankungen unterworfen waren [7].

Auch gelang es ohne einer der hier vorgestellten KIs, sondern mit logischer Programmierung ein Programm für verschiedene Brettspiele – darunter *Vier Gewinnt* – zu entwickeln. Dies lernte aus aufgezeichneten Spielen nicht nur, welche Züge eine gute Wahl sind, sondern auch die Regeln des Spieles selbst [4].

3 Die Trainingsdaten

Für das Training der KIs wurde das *Connect-4 Data Set* aus dem *UCI Machine Learning Repository* [1] verwendet. Diese bestehen aus 43 Attributen, wobei das letzte Attribut die Klasse angibt. Die Attribute 1 bis 42 entsprechen dem Spielfeld und können die Werte b für ein blankes Feld, x für den ersten Spieler oder o für den zweiten Spieler haben. Die Anzahl an gelegten Steinen der jeweiligen Spieler (x und o) ist dabei in jedem Eintrag stets gleich häufig vertreten. Die Klasse steht mit *win*, *draw* oder *loss* dafür, wie das Spiel bei perfekter Taktik für denjenigen Spieler ausgeht, der als nächstes am Zug ist. Ein für die KI gutes Ergebnis wäre somit *loss*, da es impliziert, dass der menschliche Spieler verliert. Insgesamt zählt das Trainingsset 67557 Daten. Ein Eintrag sieht zum Beispiel wie folgt aus:

```
x,b,b,b,b,b,b,b,b,b,b,b,x,o,o,x,
b,b,o,b,b,b,b,b,b,b,b,b,b,x,b,b,
b,b,b,o,b,b,b,b,b,loss
```

Es sind jedoch nur Situationen beinhaltet, in denen der nächste Zug nicht erzwungen ist, sprich: In denen der Spieler keine Dreierkette des Gegenspielers oder von sich selbst abschließen muss. Entsprechend ist nicht damit zu rechnen, dass die KIs in diesen Situationen korrekt handeln würden. In solchen Fällen wird daher die Entscheidung nicht der KI überlassen, sondern automatisch getroffen. Dafür wird von jeder Position auf dem Spielbrett ausgehend überprüft, ob entweder der menschliche Gegenspieler oder die KI im nächsten Zug vier Steine verbinden kann. Kann die KI gewinnen, wird diese Möglichkeit gewählt, sollte eine unmittelbare Gefahr durch den Gegenspieler bestehen, wird dies verhindert, und ansonsten die KI für den nächsten Zug befragt.

Für das Trainieren von Multilayer Perceptron und Radial Basis Function Network mussten die Zeichen für die Steine und Wörter für die Ziele mit Zahlenwerten ersetzt werden. Freie Felder erhielten den Wert 0.0, Steine des menschlichen Spielers den Wert 1.0 und Steine der KI den Wert -1.0. Das Ziel *win* wurde mit 0.0 belegt, *draw* mit 0.5 und *loss* mit 1.0.

4 Künstliche Intelligenzen

Zum Vergleich wurden verschiedene Typen an Künstlichen Intelligenzen trainiert. Dabei wurde ausschließlich ein überwachtes Lernverfahren eingesetzt. Dies bedeutet, die Trainingsdaten enthalten neben den Eingabewerten auch bereits die zu erzielende Klassifizierung, so dass

die KI ihr Ergebnis mit dem gewünschten vergleichen kann und so daraus lernt.

Das Multilayer Perceptron ist ein Netzwerk mit in der Regel drei Schichten von Neuronen, wobei die Neuronen benachbarter Schichten miteinander verbunden sind. Lernen erfolgt indem diese Verbindungen unterschiedlich gewichtet werden.

Das Radial Basis Function Network besitzt ebenfalls Neuronen mit gewichteten Verbindungen, basiert jedoch auf der Positionierung dieser Knoten im Eingaberaum. Nahe beieinander liegende Neuronen reagieren dabei ähnlich stark auf die gleichen Eingaben.

Der Decision Tree ist ein anderer Ansatz, bei dem eine Baumstruktur aufgebaut wird. Die Attribute der Trainingsdaten sind die Knoten, ihre möglichen Ausprägungen die Zweige und die Blätter liefern die Klassifizierung.

4.1 Multilayer Perceptron

Das Multilayer Perceptron ist eine erweiterte Variante des sogenannten Perceptrons und besitzt neben den dort vorzufindenden Ein- und Ausgabeneuronen noch eine zusätzliche versteckte Schicht an Neuronen. Von den Neuronen einer Schicht führen Verbindungen zu den Neuronen der Folgeschicht. Diese Verbindungen sind gewichtet, was den zentralen Kern der Intelligenz ausmacht. Durch das Gewicht einer Verbindung wird einem darüber geleiteten Wert eine unterschiedliche Bedeutung beigemessen.

Beim Training werden die Daten zuerst vorwärts durch das Netzwerk geleitet und für jedes Neuron wird die Aktivierung berechnet. Für ein Neuron j in der versteckten Schicht werden zunächst alle Gewichte v_{ij} der zu j führenden Verbindungen mit den dort anliegenden Eingabewerten x_i multipliziert und anschließend alle aufsummiert (siehe Formel 4.1, vgl. [6]).

$$h_j = \sum_i x_i v_{ij} \tag{4.1}$$

Dieses berechnete h_j wird für die Aktivierungsfunktion (Formel 4.2) benötigt. β (beta) ist ein frei wählbarer, positiver Faktor, für den aber meistens der Wert 1 genommen wird.

$$a_j = g(h_j) = \frac{1}{1 + \exp(-\beta h_j)} \tag{4.2}$$

Für die Neuronen der Ausgabeschicht gilt das gleiche Verfahren, jedoch werden nicht die ursprünglichen Eingabewerte x_i genommen, sondern die Aktivierungswerte a_j der Neuronen der versteckten Schicht und ihre entsprechenden Gewichtungen w_{jk}. Das Ergebnis des Vorwärtsprozesses sind dann die Werte y_k.

$$h_k = \sum_j a_j w_{jk} \tag{4.3}$$

$$y_k = g(h_k) = \frac{1}{1 + \exp(-\beta h_k)} \tag{4.4}$$

Wurden nun auch die Aktivierungen der Ausgabeschicht berechnet, wird dieses Ergebnis mit den vorgegebenen Zielwerten verglichen. Es wird der Fehler zwischen Ergebnis und Ziel errechnet und rückwärts durch das Netzwerk geführt (*Backpropagation*), was bedeutet, die Gewichte anzupassen.

Erst wird der Fehlerwert δ_{ok} (delta) in der Ausgabeschicht berechnet, wie in Formel 4.5 gezeigt. Die t_k sind die von den Trainingsdaten vorgegebenen Zielwerte, die für die jeweilige Eingabe getroffen werden sollen, was eine korrekte Klassifizierung bedeuten würde.

$$\delta_{ok} = (t_k - y_k) y_k (1 - y_k) \tag{4.5}$$

Im nächsten Schritt kann dann der Fehler δ_{hj} der versteckten Schicht berechnet werden (Formel 4.6).

$$\delta_{hj} = a_j(1 - a_j) \sum_k w_{jk}\delta_{ok} \tag{4.6}$$

Nun können die Gewichte angepasst werden. Dafür wird der Fehler mit dem Eingabewert des Neurons multipliziert und auf die Gewichtung addiert. Um das Lerntempo zu beeinflussen kann zudem noch der Faktor η (eta) angepasst werden. Formel 4.7 zeigt das Vorgehen für die Ausgabeschicht und Formel 4.8 entsprechend für die versteckte Schicht.

$$w_{jk} = w_{jk} + \eta\delta_{ok}a_j^{hidden} \tag{4.7}$$

$$v_{ij} = v_{ij} + \eta\delta_{hj}x_i \tag{4.8}$$

Danach kann die nächste Lernphase beginnen oder das Training beendet werden. Wenn nun aber alle Eingabewerte 0 wären, hätten die Gewichtungen keine Bedeutung, da Eingabe und Gewicht anfangs mulipliziert werden. Um trotzdem das Netzwerk in diesem Fall trainieren zu können, wird den Schichten jeweils ein weiteres Neuron hinzugefügt, das sogenannte *Bias-Neuron*. Das Bias-Neuron hat immer den Wert -1, verhält sich aber sonst wie die anderen Neuronen seiner Schicht auch. Während dem Training kann die Gewichtung seiner Verbindung also angepasst werden, um die Ausgabe zu beeinflussen [6]. Abbildung 4.1 zeigt noch einmal den Aufbau eines MLP inklusive eines Bias-Neurons in der versteckten Schicht.

Das hier verwendete MLP hatte 42 Eingabeknoten, 40 Knoten in der versteckten Schicht und einem Ausgabeknoten, sowie jeweils einem Bias-Neuron in Eingabe- und versteckter Schicht.

Als Trainingsverfahren wurde *early stopping* gewählt. Dabei wird der Lernalgorithmus wiederholt angewendet, bis ein berechneter Fehler zwischen zwei Iterationen einen vorgegebenen Schwellenwert unterschreitet. Dieser Fehler wird berechnet, indem zunächst von dem Trainingsset eine deutlich kleinere Untermenge

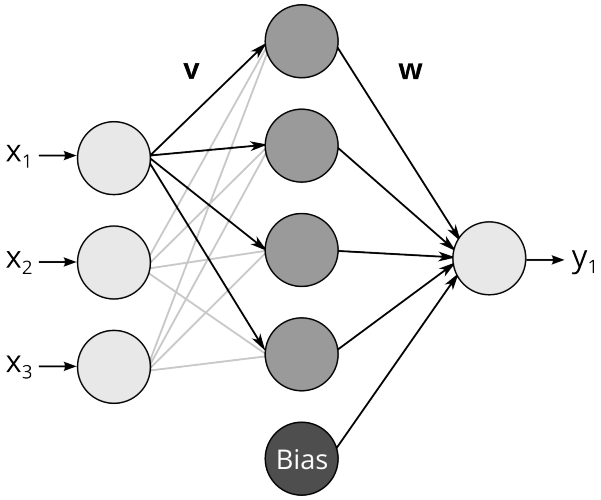

Abbildung 4.1: Aufbau eines MLPs. Der Eingabevektor $\mathbf{x}$ wird von links aus durch das Netzwerk geleitet und führt zur Ausgabe $\mathbf{y}$.

an Trainingsdaten herausgenommen wird. Die Daten werden zur Validierung der Lernergebnisse verwendet. Wenn nun das MLP einmal trainiert wurde (eine Vorwärtsphase und einmal Backpropagation durchlaufen hat), werden die Trainingsdaten darauf angewandt und der Fehler zwischen Ergebnis und Zielwert berechnet.

Der Vorteil von *early stopping* besteht darin, dass es unwahrscheinlicher ist, dass ein *overfitting* der KI eintritt, da die Validierungsdaten nicht mehr Teil der Trainingsdaten sind. *Overfitting* beschreibt das Problem, dass eine KI sich zu sehr den Trainingsdaten anpasst und nur für diese gute Ergebnisse liefert. Wird eine solche KI später mit validen, aber nicht in den Trainingsdaten beinhalteten Eingabewerten konfrontiert, liefert sie in dem Fall schwache Ergebnisse.

Hier wurde der Wert 1 für besagte Fehlerdifferenz genommen. Trainiert wurde das MLP mit allen 67557 Datensätzen über 40 Iterationen je Trainingsrunde, mit einer Lernrate η von 0.35 und einem Momentum von 0.7. Die Aktivierungsfunktion war linear. In Testspie-

len zeigte sich, dass Training mit weniger Datensätzen als dem vollen Set zu schlechteren Spielleistungen des MLP führte.

Bei der späteren Benutzung wird dem MLP das komplette Spielbrett in der gleichen Form wie ein Eintrag aus den Trainingsdaten übergeben. Die Ausgabe ist in der Regel ein Gleitkommawert zwischen 0.0 und 1.0 – wobei auch Ausreiser möglich sind – entsprechend den Zielwerten von 0.0 für *win*, 0.5 für *draw* und 1.0 für *loss*. Im nächsten Schritt wird berechnet, zu welchem Ziel die Ausgabe am nächsten steht und als dieses Ziel eingestuft. Dies wird für alle verfügbaren Spalten durchgeführt und am Schluss diejenige gewählt, die den besten weiteren Verlauf verspricht.

Dieses trainierte MLP zeigte jedoch eine Schwäche innerhalb der ersten Züge des Spiels. Denn wählte der menschliche Spieler den mittleren Bereich des Spielbrettes, also die Spalten 3, 4 oder 5, wählte das MLP dieselbe Spalte. Dies führte dazu, dass der menschliche Spieler innerhalb von vier Zügen direkt gewinnen konnte. Um dies zu verhindern, wurde ein kleiner Workaround eingeführt, der der KI verbietet, bei ihrem allerersten Zug dieselbe Spalte wie der menschliche Gegenspieler zu nehmen.

Da bei kleineren beziehungsweise größeren Spielbrettern die Anzahl der Eingabewerte – entsprechend der Anzahl an Feldern – nicht mit der der Eingabeneuronen und deren Verbindungen übereinstimmt, wurde sich dafür entschieden, bei mehr Werten als Neuronen, neue Verbindungen mit einem Wert von 0.5 hinzuzunehmen. Bei weniger Werten werden nicht alle Verbindungen berücksichtigt und entsprechend ihre Aktivierung nicht berechnet.

4.2 Radial Basis Function Network

Die Idee beim Radial Basis Function Network (RBF Network) ist, dass die Neuronen für Eingabewerte, die sich ähneln, auch ähnlich stark feuern. Man bezeichnet die Neuronen daher auch als „lokale Neuronen“, da sie sich für bestimmte Bereiche im Eingaberaum spezialisieren.

Diese „lokale“ Eigenschaft der Neuronen wird durch die gewählte Aktivierungsfunktion bewerkstelligt. Dabei kommt eine leicht veränderte normalisierte Gauß'sche Funktion zum Einsatz:

$$g(\mathbf{x}, \mathbf{w}, \sigma) = \frac{\exp\left(\frac{-||\mathbf{x}-\mathbf{w}||}{2\sigma^2}\right)}{\sum_i \exp\left(\frac{-||\mathbf{x}-\mathbf{w}_i||}{2\sigma^2}\right)} \tag{4.9}$$

Dabei steht $\mathbf{x}$ für den Eingabevektor, $\mathbf{w}$ sind die Gewichtungen und σ (sigma) ist ein Faktor, der die Breite des Gauß'schen Feldes beeinflusst. Wird das σ zu groß gewählt, reagiert das Neuron auf alle möglichen Eingaben, anstatt sich auf einen lokalen Bereich zu beschränken. Ist der Wert hingegen zu klein, erkennt das Neuron passende, aber leicht verrauschte Eingaben nicht mehr und spricht auch auf ähnliche Eingabewerte nicht mehr gut an. Wie man sich die Felder um die Neuronen vorstellen kann, ist in Abbildung 4.2 zu sehen. Ähnlich dem MLP können diese Neuronen als eine versteckte Schicht betrachtet werden.

Die Gauß'schen Felder sollten möglichst viel Raum abdecken, um auf die möglichen Eingaben reagieren zu können. Das bedeutet, dass es am besten wäre, bei wenig Neuronen ein größeres σ zu wählen und bei vielen Neuronen ein kleineres. Entsprechend feiner bzw. gröber reagiert das Netzwerk dann auch auf die Eingaben. Ein Weg, zu Beginn σ bestimmen zu lassen, ist mit Formel 4.10 beschrieben. M ist die gewählte Anzahl an RBF-Neuronen und d der maximale euklidische Abstand zwischen ihnen.

$$\sigma = d/\sqrt{2M} \tag{4.10}$$

Der maximale euklidische Abstand d für n-

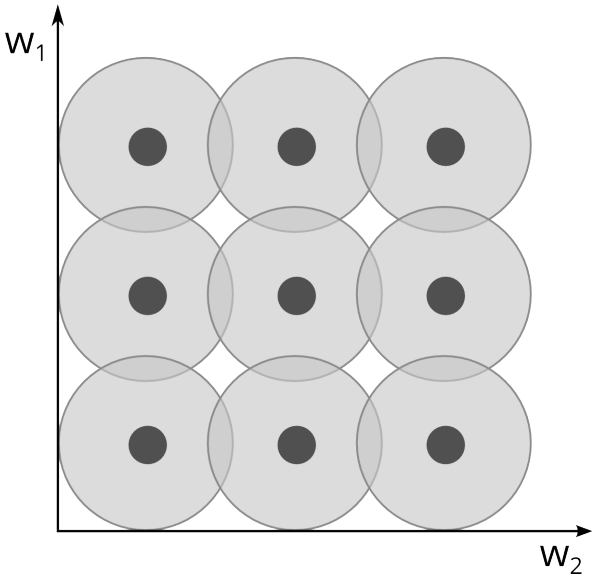

Abbildung 4.2: Die Neuronen eines RBFs dargestellt im Gewichtsraum. Die durch σ bestimmten Felder um die Neuronen herum können sich, wie hier zu sehen, auch überlappen.

dimensionale Eingaben berechnet sich über folgenden Algorithmus:

1. Finde für jeden Eintrag in den Trainingsdaten jeweils den maximalen Wert x_max.
2. Finde für jeden Eintrag in den Trainingsdaten jeweils den minimalen Wert x_min.
3. Ziehe jeweils das Minimum vom Maximum ab. Sei dies der Wert x_diff.
4. Der maximale euklidische Abstand d ist dann das größte dieser x_diff.

Die Neuronen in der Ausgabeschicht können von anderer Natur sein. In dieser Arbeit wurde dafür das „McCulloch and Pitts“-Neuron verwendet, welches als Ausgabewerte für einen bestimmten Schwellenwert nur *feuern* und *nicht feuern* kennt. Es handelt sich hierbei um ein Perceptron. Normalerweise wird als Schwellenwert für das Feuern des Perceptrons der Wert 0 genommen. Liegt der Eingabewert darüber, wird gefeuert (Ausgabe 1), liegt er darunter, wird nicht gefeuert (Ausgabe 0). Da sich nach erfolgtem Training jedoch zeigte, dass das Perceptron sehr häufig feuerte, wurde es angepasst. Die Eingabewerte werden zunächst auf den Intervall -1.0 bis $+1.0$ normalisiert und als Schwellenwert wird nun 0.5 herangezogen.

Für das Training müssen zuerst die RBF-Neuronen positioniert werden. Die initialen Positionen können entweder zufällig vergeben oder mit dem K-Means-Algorithmus intelligent gewählt werden. K-Means analysiert dabei die Trainingsdaten, um Cluster – Häufungen sich ähnelnder Daten – zu finden [5]. Dabei wird wie folgt vorgegangen:

1. Die Daten werden bereitgestellt.
2. Die Anzahl k an zu erstellenden Cluster-Zentren wird vorgegeben.
3. Es werden k viele Vektoren (genannt *Codebookvektoren*) zufällig gewählt.
4. Jeder Datenpunkt wird dem nächsten Codebookvektor zugeordnet.
5. Für die so entstandenen Cluster wird der jeweilige Schwerpunkt ermittelt.
6. Die Codebookvektoren werden auf die Cluster-Schwerpunkte platziert.
7. Wiederhole ab Schritt 4, bis keine Änderungen mehr eintreten.

Wurden die RBF-Neuronen positioniert, wird im nächsten Schritt für die Trainingsdaten die Aktivierung in den RBF-Neuronen mittels Formel 4.9 berechnet. Die Ergebniswerte der Aktivierungen werden als Eingaben zum Training an das Perceptron weitergereicht. Danach ist das RBF-Netzwerk einsatzbereit [6].

Als Trainings-Parameter wurden hier ein η von 0.4, 30 RBF-Neuronen und 40 Iterationen

im Perceptron gewählt. K-Means wurde eingesetzt und die Ausgaben der RBF-Neuronen vor der Weitergabe an das Perceptron normalisiert. Der Wert für σ wurde automatisch bestimmt und beträgt 0.258199.

Zur Benutzung wird dem RBF Network ein Spielbrett in der Form übergeben, wie auch die Einträge in den Trainingsdaten aussehen. Die Ausgabe ist ein Tripel, wobei der erste Wert die Klassifikation als *loss* – also Verlust des menschlichen Spielers – ist, der zweite als *draw* und der dritte als *win*. Die Einstufung erfolgt nur als „gehört zu der Klasse“ oder „gehört nicht dazu“. Dabei kann es auch sein, dass die Eingabe mehr als einer Klasse zugeordnet wird. Bei der Wahl einer Spalte anhand der Klassifizierungen wurde der Hauptfokus auf die Einstufung als *loss* gelegt. Leider tendiert das RBF Network dazu, mehrere Spalten gleich zu klassifizieren, was eine Auswahl der besten Strategie erschwert. Dies führt dazu, dass bei Präsentation der Spalten in der immer gleichen Reihenfolge, immer die zuletzt geprüfte Spalte gewählt wird. Um die Wahl zumindest dem menschlichen Spieler gegenüber etwas geplanter erscheinen zu lassen, wird daher aus den best-klassifizierten Spalten zufällig eine ausgewählt.

4.3 Decision Tree

Bei einem Decision Tree (DTree) wird jede Eingabe als eine Menge von Attributen betrachtet, die den Zweigen eines Baumes entsprechen. Das Blatt des letzten Zweiges für die gegebene Eingabe steht schließlich für die Ausgabe der KI.

Das Training, oder vielmehr der Aufbau des Baumes, besteht darin, für die jeweilige Tiefe im Baum das geeignete Attribut zu finden, das als Verzweigung und somit als Entscheidungspunkt verwendet wird. Dafür betrachtet man die *Entropie*, das heißt den Informationsgehalt der verschiedenen Attribute. Formel 4.11 errechnet die Entropie H eines Attributes p aus den Wahrscheinlichkeiten mit denen das Attribut zu einem bestimmten Ergebnis führt.

$$H(p) = -\sum_i p_i \log_2 p_i \tag{4.11}$$

Darauf aufbauend wird im DTree der *ID3*-Algorithmus eingesetzt. Das Ziel dabei ist es, das Attribut für den nächsten Knoten anhand dessen zu wählen, wie sehr es die Entropie des gesamten Trainingssets senken würde. Dafür betrachtet man den Informationsgewinn (engl. *information gain*), der definiert ist als die Gesamtentropie des Sets minus die Entropie eines gewählten Attributes. Formel 4.12 beschreibt dies, wobei $H(S)$ die Entropie der Trainingsdaten (inklusive der Zielwerte) ist, F ist das gewählte Attribut, f sind alle möglichen Ausprägungen für das Attribut und S_f ist die Menge der Trainingsdaten ohne das Attribut F.

$$Gain(S, F) = H(S) - \sum_{f \in values(F)} \frac{|S_f|}{|S|} H(S_f) \tag{4.12}$$

Der Informationsgewinn wird für jedes Attribut errechnet und im Anschluss jenes gewählt, das den höchsten Wert liefert. Die diversen möglichen Ausprägungen des Attributs bilden dabei die Verzweigungen, die vom Knoten ausgehen. Für die darauffolgenden Knoten beginnt die Berechnung wieder von vorne, diesmal jedoch ohne das bereits verwendete Attribut.

Wenn man später den DTree verwenden möchte, übergibt man ihm als Eingabe eine Zuordnung von Attribut zu Wert – im Falle von *Vier Gewinnt* eine Bezeichnung für jedes Feld und dessen Belegung (Stein des Spielers oder ob das Feld noch frei ist) – woraufhin der Baum durchlaufen wird. Die Rückgabe ist dann das erreichte Blatt im Baum.

Im Fall von *Vier Gewinnt* waren die Attribute die Bezeichnungen der einzelnen Felder.

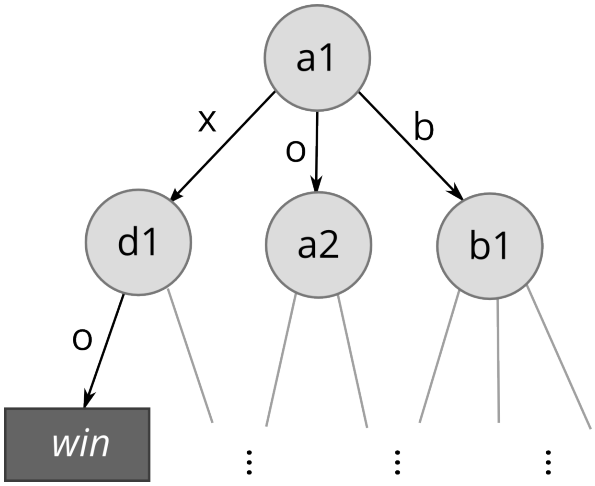

Abbildung 4.3: Möglicher Aufbau eines DTrees. *a1* entspricht dem Feld in der ersten Spalte und der ersten Zeile. Befindet sich dort ein Stein des Menschen, folgt man dem Zweig *x*, für einen KI-Stein dem Zweig *o* und ansonsten dem Zweig *b*. *win* ist ein zu erreichendes Blatt.

Ihre zugeordneten Werte waren die Belegungen – der Stein welches Spielers sich darauf befand oder ob das Feld frei war (siehe Abbildung 4.3 für ein Beispiel). Die Rückgabe war entsprechend der Trainingsdaten entweder *loss*, *draw* oder *win*. Zusätzlich musste noch der Fall abgefangen werden, wenn für eine Belegungsabfolge kein Pfad im Baum existierte. In diesem Fall wurde *unknown* zurückgegeben. Dies ist zum Beispiel der Fall bei größeren Spielbrettern, da Spalten oder Zeilen hinzukommen, für die der DTree keine Knoten besitzt.

Wie zuvor auch, wird der DTree für alle noch wählbaren Spalten befragt. Jene mit dem Ergebnis *loss* haben eine hohe Priorität, da sie die Niederlage des menschlichen Spielers anzeigen, während *win* die niedrigste Priorität hat. Selbst *unknown* sollte noch bevorzugt werden, da ein unbekanntes Auskommen immer noch besser ist, als eine vorhergesagte Niederlage der KI.

Der DTree tendiert – wie schon das RBF – dazu, mehreren Spalten die gleich-höchste Priorität zukommen zu lassen und so beispielsweise wiederholt die letzte Spalte zu wählen. Um die Spaltenwahl etwas abwechslungsreicher zu gestalten, wird daher zufällig eine Spalte aus den am besten eingeschätzten gewählt.

5 Evaluation

Für die Evaluation wurde eine webbasierte Benutzeroberfläche zum Spielen in HTML und JavaScript realisiert (siehe Abbildung 5). Spieler entscheiden zuerst, gegen welchen KI-Typen sie spielen möchten, und wählen dann auf einer grafischen Repräsentation des Spielbrettes nur noch abwechselnd mit der KI ihre Spalten aus, in der ihr Spielstein geworfen werden soll.

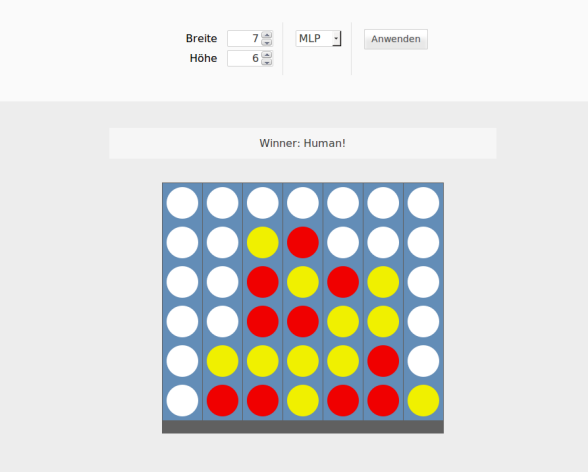

Abbildung 5.1: Web-Oberfläche zum Spielen gegen eine vortrainierte KI.

5.1 Standard-Spielbrett 7x6

Auf einem Spielbrett der standardmäßigen Größe von 7x6 Feldern, auf dem auch die Trainingsdaten basieren, beweist sich das MLP als guter Gegner. Sowohl gegen erfahrene als auch weniger erfahrene menschliche Gegenspieler hat es gewinnen können. Jedoch wiederho-

len sich spätere Spiele, da das MLP für eine bestimmte Spielbrettbelegung immer dieselbe Spalte wählt. Dies liegt an der feinen Ausgabe (Gleitkommazahlen mit mehr als fünf Nachkommastellen) des MLP, die es sehr unwahrscheinlich macht, dass zwei oder mehr Spalten während eines Zuges die gleiche Priorität erhalten und somit keinen Raum für eine zufällige Wahl bieten.

Das RBF und der DTree sind beide leider sehr schwache Spieler. Ihre Züge wirken oft rein nach Zufall gewählt, was zum Teil auch zutreffen dürfte, da aus den am höchsten priorisiertesten Spalten zufällig eine gewählt wird. Im Gegensatz zum MLP beschränken sich die Ausgabewerte auch nur auf *1/0* bzw. *loss/draw/win*, was eine feinere Auswahl der Spalten nicht ermöglicht.

5.2 Kleineres Spielbrett

Auf kleineren Spielbrettern – sei es in Breite, Höhe oder beidem – schwächelt das MLP. Man hat durchaus bisweilen noch den Eindruck, hinter den Zügen stecke eine Taktik, aber es begeht häufig Fehler, die im nächsten Zug zum Sieg des menschlichen Spielers führen.

RBF und DTree sind weiterhin sehr schwache Spieler und sind als schlechter einzustufen als das MLP.

5.3 Größeres Spielbrett

Auf größeren Spielbrettern – sei es in Breite, Höhe oder beidem – zeigt das MLP ebenfalls Schwächen. Jedoch wirken die Züge oft durchdacht in der Hinsicht, dass auf einen Sieg hingearbeitet wird. Die Leistung ist schlechter als auf einem Standard-Brett, aber besser als auf kleineren Brettern.

Als sehr schwache Spieler erweisen sich wieder RBF und DTree, die sich mit Leichtigkeit vom menschlichen Spieler austricksen lassen. Speziell beim DTree ist dies jedoch nicht überraschend, da er Werte für Spalten oder Zeilen erhält, die in seiner Struktur schlichtweg nicht existieren. Damit verkommt die Spaltenwahl zu einer Zufallswahl.

6 Fazit

Ziel war es, einer Künstlichen Intelligenz das Brettspiel *Vier Gewinnt* beizubringen. Dafür wurden verschiedene Typen betrachtet: Das Multilayer Perceptron (MLP), das Radial Basis Function Network (RBF Network) und der Decision Tree (DTree). Alle drei wurden mit dem vollen Trainingsset in einem überwachten Verfahren trainiert und im Spiel gegen menschliche Spieler getestet. Dabei zeigte sich, dass das RBF Network und der DTree untauglich waren, während das MLP jedoch einen sehr guten Gegenspieler abgab. Bei Veränderung des Spielbrettes zu einer anderen Größe als der trainierten, zeigte es jedoch leichte Schwächen.

Dieses Ergebnis sagt jedoch nicht per se aus, dass das RBF Network oder der DTree nicht für das Spiel geeignet seien. Es gibt viele Faktoren, die einen Einfluss auf das Ergebnis haben. Vielleicht wurde eine gute Parameter-Kombination nicht entdeckt, ein anderes Format der Trainingsdaten inklusive Klasse hätte besser funktioniert oder ein Algorithmus ließe sich abändern. Auch könnte man beim Training noch unterstützende Verfahren wie Boosting oder Bagging hinzunehmen, um zum Beispiel beim MLP unter Umständen noch eine Leistungsverbesserung zu beobachten.

Hier wurde zudem nur mit überwachten Lernverfahren gearbeitet. Es könnten noch unüberwachte Verfahren geprüft werden oder Online-Learning, bei dem die KI mit jedem Spiel weiter dazulernt.

Was das Ergebnis auf jeden Fall gezeigt hat, ist, dass Multilayer Perceptrons für *Vier Gewinnt* geeignet sind.

Quellen

[1] UCI Machine Learning Repository: Connect-4 Data Set. http://archive.ics.uci.edu/ml/datasets/Connect-4. [letzter Zugriff: 2012-06-16].

[2] Allen Brubaker, Jason Long. The Connect Four Artificial Player. http://code.google.com/p/connect4neuralnetwork/, 2010.

[3] Victor Allis. A knowledge-based approach to connect-four. the game is solved: White wins. In *Master's thesis, Vrije Universiteit*, page 4, 1988.

[4] Łukasz Kaiser. Learning games from videos guided by descriptive complexity. In *Proceedings of the 26th Conference on Artificial Intelligence, AAAI-12. To appear.* AAAI Press, 2012.

[5] David Kriesel. *Ein kleiner Überblick über Neuronale Netze.* 2007. erhältlich auf http://www.dkriesel.com.

[6] Marsland, Stephen. *Machine Learning: An Algorithmic Perspective.* Chapman & Hall/CRC, 1st edition, 2009.

[7] Marvin Oliver Schneider and João Luís Garcia Rosa. Neural Connect 4 - A Connectionist Approach to the Game. In *Proceedings of the VII Brazilian Symposium on Neural Networks (SBRN'02)*, SBRN '02, Washington, DC, USA, 2002. IEEE Computer Society.