Temporal Difference Learning for Game Playing

Wenchao Li

Hauptseminar *Neuroinformatik*
Universität Ulm, Abteilung Neuroinformatik, Sommersemester 2007

Zusammenfassung

Heutige Schachprogramme sind in der Lage den amtierenden Weltmeister zu besiegen. In dieser Seminararbeit wird vorgestellt, wie Temporal Difference (TD) Lernen für Schachprogramme bis jetzt verwendet wird. Ziel dieser Seminararbeit ist es zu zeigen, dass die Spielprogramme mit Hilfe von TD Lernen und schon jetzt das Niveau sehr guter menschlicher Spieler erreicht haben. Im ersten Teil werden zwei bekannte Programme TD-Gammon und KnightCap als Beispiele beschrieben. Im zweiten Teil wird die Arbeit von Jonathan Schaeffer, Marikian Hlynka und Vili Jussila betrachtet und untersucht, ob die Gewichtung der Bewertungsfunktion von Chinook durch den TD-Leaf Algorithmus abgestimmt werden kann.

Key words: temporal difference(TD), evaluation function(EvalFunc)

1 Einleitung

Im Jahr 1992 gelang es Tesauro mit neuronalen Netzen als Stellungsbewertungsfunktionen eines der stärksten Backgammon-Programme zu entwickeln (TD-Gammon genannt). Dieses ist der bis jetzt größte Erfolg bei der Verwendung von neuronalen Netzen in Spielprogrammen. Der dabei verwendete Trainingsalgorithmus war der TD(λ)-Algorithmus von Sutton. Das Problem bei diesem Algorithmus ist, dass nicht alle Spiele die gleichen Bedingungen aufweisen und somit diese Formel nicht auf alle Spiele übertragbar ist.

Vom Erfolg des Tesauros Backgammon-Programmes beeindruckt, haben Jonathan Baxter, Andrew Tridgell und Lex Weaver versucht die gleiche Lernstrategie in ein Schachprogramm, nämlich KnightCap, umzusetzen. Der TDLeaf-Algorithmus wurde als Trainingsalgorithmus verwendet. Im folgenden Abschnitt werden die beiden Programme detailliert beschrieben.

2 TD-Gammon

TD-Lernen, ist zuerst von Samuel (1959) entwickelt und später von Sutton(1988) mit TD(λ) erweitert und formalisiert worden. TD-Lernen ist eine Kombination aus der Monte-Carlo-Idee und der dynamischen Programmierung(DP). Wie das Monte-Carlo-Verfahren kann das TD-Verfahren direkt mit grober Erfahrung ohne ein Modell der Umweltdynamik

lernen. Der Vorteil von TD-Lernen ist aber, dass nicht wie bei Monte-Carlo-Verfahren der Zyklus einmal durchlaufen werden muss, um die Parameter zu aktualisieren. Es kann schon im nächsten Zustand eine Aktualisierung des Vorhergehenden vollzogen werden. Trotzdem konvergiert TD-Lernen zu einer optimalen Strategie. Wie Dynamische Programmierung, updated TD die Zustandswerte anhand bereits gelernter Zustände.

Um die Effektivität von TD-Lernen in einem Schachprogramm zu testen, wurde diese im Schachprogramm integriert, wie er auch bei dem erfolgreichen TD-Gammon (von Tesauro) angewendet wurde. Das Herzstück der TD-Gammon ist ein neuronales Netzwerk mit einer Multiplayer Perzeptron (MLP)-Architektur. In TD-Gammon wurde eine Kombination aus dem TD(λ)-Verfahren und einer nichtlinearen Funktionsapproximation benutzt.

Der TD-Gammon trainiert sich folgendermaßen: Das Netzwerk überwacht immer die Stellungssequenzen der Spielsteine. Die Brettposition werden als Eingabevektor x[1], x[2]... x[n] in den Netzwerk kodiert und für jede Eingabe x[t] gibt es ein Ausgabevektor Y[t], um das erwartete Ergebnis von x[t] anzuzeigen. Der TD(λ) Algorithmus wird hier als Update-Formeln für die Gewichte angewendet: (Vergleich mit Formeln (2)):

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k \tag{1}$$

Die Bedeutungen aller Parameter werden im nächsten Abschnitt detailliert angegeben. Am Spielende wird ein Reward z zurückgeliefert, um die Gewichtung zu adaptieren. Weil die TD-trainierte Netzwerke mit Eingabekodierung ohne Erfahrungen gleichstark wie Neurogammon spielt (ein Vorläufer von TD-Gammon und gleichzeitig Olympiasieger in Backgammon von 1989), haben die Entwickler einige durch einer Experten eingestellten Feature von Neurogammon in TD-Gammon eingesetzt. Dadurch stieg die Spielstärke von TD-Gammon über Neurogammon und allen anderen Backgammon-Programme.

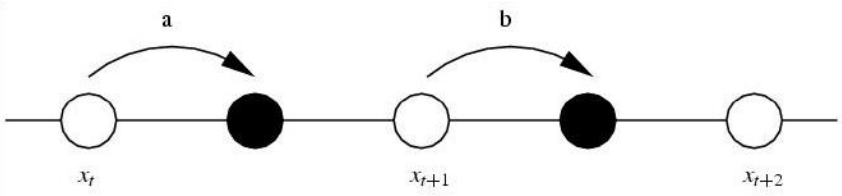
Program	Training Games	Opponent	Results
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0,25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0,18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pts/40 games (-0,02 ppg)

Die obere Tabelle zeigt die Ergebnisse auf den Testspielen (TD-Gammon Generation vs. die Weltmeister). TDG 1.0 benutzt 1-schichtige Neuronalesnetz. TDG 2.0 und 2.1 benut-

zen 2-schichtige Netze. TDG 1.0 und 2.1 haben 80 versteckte Einheiten und 2.0 hat 40. Im Jahr 1991 hat TDG1.0 nach 51 Spielen 13 Punkte verloren. Die neuere Version TDG2.1 verlor nur 1 Punkt nach 40 Spielen gegen Robertie. Robertie war der Ansicht (nach 40 Spielen), dass TD-Gammon 2.1 ähnlich stark wie ein Weltmeister sein konnte. Er glaubte, die geringen Probleme seien technische Fehler.

3 Temporale Differenz und TD (λ)

TD(λ) ist eine elegante Approximationstechnik. Die Parameter werden online, nach jedem Zustandsübergang oder möglicherweise in einem Stapel von Anpassungen, nach einigen Übergängen, angepasst. Der TD-Algorithmus wird nun aus der Sicht eines Agenten diskutiert.



Sei S die Menge aller möglichen Brettkombinationen. Spielvorgänge mit einer Zugserie zu einer diskreten Zeit werden schrittweise mit t = 1,2, ... angegeben. Zum Zeitpunkt t befindet sich der Agent auf dem Brett $x_t \in S$ und besitzt eine Liste von Zugmöglichkeiten (*actions*) A_{x_t} (als Aktion gilt, jeder legale Zug in der Stellung x_t). Der Agent wählt nun eine Aktion $a \in A_{x_t}$ aus und damit den Übergang des Zustands x_t zu x_{t+1} mit der Wahrscheinlichkeit $p(x_t, x_{t+1}, a)$. Hier ist x_{t+1} die Brettposition, nachdem der Agent einen Zug gemacht hat und der Gegner die Antwort gibt. Nachdem das Spiel zu Ende ist, bekommt der Agent einen Wert zurück (Reward), beispielsweise 0 für Remis, 1 für Sieg und -1 für eine Niederlage. Wir nehmen an, dass jedes Spiel genau eine fixe Länge von N Zügen besitzt. Nun sei $r(x_N)$ der Reward.
Betrachten wir die ideale Bewertungsfunktion $J^*(x)$.

$$J^*(x) := E_{x_N|x} r(x_N)$$

wobei das der Erwartungswert der Belohnung (*reward*) am Ende der Partie ist. Es ist schwer anzunehmen, dass jede Schachstellung durch eine lineare Bewertungsfunktion genau bewertet werden kann. Das Ziel ist nun, die ideale Bewertungsfunktion $J^* : S \mapsto \mathbb{R}$ durch eine lineare Funktion zu approximieren. Wir betrachten dazu eine parametrisierte Klasse von linearen Funktionen $\tilde{J}(\cdot, \omega) : S \times \mathbb{R}^k \mapsto \mathbb{R}$. Gesucht ist $\omega = (\omega_1, ..., \omega_k)$ der Satz von Parametern, der die ideale Bewertungsfunktion J^* möglichst gut approximiert. Der TD(λ)-Algorithmus tut nun genau das.

Sei $x_1, ..., x_{N-1}, x_N$ ist eine Zustandssequenz für eine gespielte Partie. Für einen gegebenen Vektor ω, ist die *Temporale Differenz* von $x_t \to x_{t+1}$ definiert als

$$d_t := \tilde{J}(x_{t+1}, w) - \tilde{J}(x_t, w)$$

Man merke sich, dass d_t die Differenz zwischen dem Spielausgang von $\tilde{J}(\cdot, \omega)$ zum Zeitpunkt t+1 und dem Resultat von $\tilde{J}(\cdot, \omega)$ zum Zeitpunkt t ist. Für die ideale Bewertungsfunktion J^* gilt:

$$E_{x_{t+1}|x_t}[J^*(x_{t+1}) - J^*(x_t)] = 0$$

Falls $\tilde{J}(\cdot, \omega)$ zu J^* gut approximiert, soll $E_{x_{t+1}|x_t} d_t$ gegen Null sein. Wir nehmen an, dass immer gilt: $\tilde{J}(x_N, \omega) = r(x_N)$, dann ergibt sich nun letztlich für die temporale Differenz

$$d_{N-1} = \tilde{J}(x_N, \omega) - \tilde{J}(x_{N-1}, \omega) = r(x_N) - \tilde{J}(x_{N-1}, \omega)$$

d_{N-1} ist die Differenz zwischen dem wahren Ausgang der Partie (*reward*) und der Voraussage zum Zeitpunkt des vorletzten Zuges. Am Ende des Spieles aktualisiert der TD(λ)-Algorithmus den Vektor ω mit der Formel

$$\omega := \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, \omega)[\sum_{j=t}^{N-1} \lambda^{j-t} d_t] \tag{2}$$

wobei $\nabla \tilde{J}(\cdot, \omega)$ der Gradient und $[\sum_{j=t}^{N-1} \lambda^{j-t} d_t] =: \Delta t$ die gewichtete Summe der Differenzen im Rest der Partie ist. Für $\Delta t > 0$ gilt, dass die Stellung x_t vermutlich unterbewertet wurde, der Vektor ω wird mit einem positiven Vielfachen des Gradienten addiert und demnach ist die Bewertung der Stellung mit den aktualisierten Parametern höher als zuvor. $\Delta t < 0$ interpretieren wir als Überbewertung der Stellung und addieren den Vektor mit einem negativen Vielfachen des Gradienten. Der positive Parameter α kontrolliert die Lernrate und konvergiert typischerweise (*langsam*) gegen 0. Der Parameter λ kontrolliert dabei den Umfang inwieweit sich die Temporale Differenz zeitlich rückwärts fortpflanzt. Um das zu sehen schauen wir uns die vorhergehende Gleichung nochmal für $\lambda = 0$ an:

$$\omega := \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, \omega) d_t = \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, \omega)[\tilde{J}(x_{t+1}, \omega) - \tilde{J}(x_t, \omega)]$$

und für $\lambda = 1$:

$$\omega := \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, \omega)[r(x_N) - \tilde{J}(x_t, \omega)]$$

Für $\lambda = 0$ wird der Parameter-Vektor so angepasst, dass $\nabla \tilde{J}(x_t, \omega)$ näher an $\nabla \tilde{J}(x_{t+1}, \omega)$ gesetzt wird. Wir beziehen uns also auf den nachfolgenden Zug. Im Kontrast dazu, TD(λ = 1) passt den Parameter-Vektor so an, dass die erwartete Belohnung zum Zeitpunkt t näher zum Resultat des Spieles zum Zeitpunkt N gesetzt wird. Ein λ-Wert zwischen 0 und 1

interpoliert zwischen diesen beiden Verhaltensweisen.

4 Minimax-Algorithmus und TD-Leaf

Wie lassen sich nun diese Erkenntnisse in einen Zug-Such-Algorithmus unterbringen? Eine Möglichkeit wäre es die gefundene Näherung $\tilde{J}(\cdot, \omega)$ zu benutzen um aus einer grossen Anzahl von Aktionen zu einer Stellung den vielversprechensten zu wählen. Dabei wird die Aktion gewählt, die anschliessend dem Gegner minimale Chancen zusprechen:

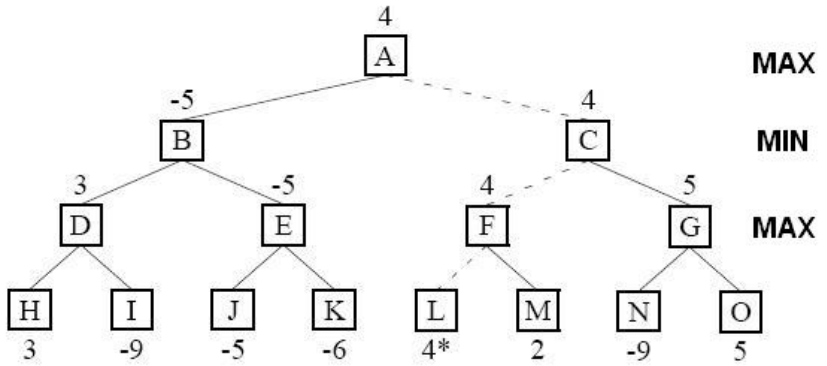
$$a^*(x) := argmin_{a \in A_x} \tilde{J}(x'_a, \omega)$$

x'_a ist die Stellung, die nach Ausführung der Aktion a in Stellung x entsteht.

Diese Strategie wird, wie bereits schon erwähnt, in Tesauros TD-Gammon verwendet, einem sehr erfolgreichen Backgammonprogramm, dass stärker als der Weltmeister spielt. Im Schach hingegen ist diese Vorgehensweise nicht sehr vorteilhaft, da doch jede Stellung sehr viele taktische Ereignisse beinhalten kann und diese nur durch einen Suchbaum identifiziert werden können. Die Idee ist, nicht auf der Wurzel des Suchbaumes zu arbeiten, sondern das am besten bewertete Blatt als Bewertungsgrundlage zu nehmen. Wir betrachten nun den `MiniMax-Algorithmus`.

In den `MiniMax-Algorithmus` wird ein vollständiger Suchbaum der Tiefe d aus den jeweiligen Zugmöglichkeiten der beiden Spieler aufgebaut. Anschliessend werden die Blätter mit einer Stellungsbewertung evaluiert. Es wird davon ausgegangen, dass jeder Spieler den für sich besten Zug spielt. Begonnen wird in der untersten Baumebene. Die Bewertung der jeweils beste Spielmöglichkeit wird an den Vaterknoten weitergeleitet. Zuletzt erhält die Wurzel einen Wert und kann nun entsprechend den ersten Zug ausführen mit der Gewissheit, dass er mindestens das forcierte Ergebnis erreichen kann.

Hilfe leistet folgend Vereinbarung: Für den weißen Spieler werden Stellungsvorteile positiv und für den Spieler mit den schwarzen Steinen negativ bewertet. Demzufolge ist Weiß am Zug bestrebt, einen möglichst großen Wert zu erhalten und wird, an welcher Stelle am Baum er auch am Zug ist, den maximalen wählen. Der Gegner wählt das Minimum usw.



Wir schauen nun die obere Abbildung: In diesem Beispiel wird der komplette Suchbaum bis zur Tiefe betrachtet. Die Stellungen in den Blattknoten H-O werden durch eine Bewertungsfunktion ausgewertet. Wir nehmen an, dass bei den Knoten der Tiefe 2 weiß am Zug

ist. Deswegen wird im linken Teilbaum der Wert 3 nach Knoten D gegeben. Analog sind die Knoten E, F, G bewertet. Danach ist schwarz am Zug, wählt er den für sich besten Zug, also -5 am Knoten B usw. Die Wurzel wird anschließend mit dem Wert 4 bewertet. Der Pfad von der Wurzel A bis zum Blatt L mit dem Wert 4 heißt auch die Hauptvariante (*Principal Variation*).

Sei $\tilde{J}_d(x, w)$ der erwartete Evaluationswert zur Stellung x, der durch $\tilde{J}(\cdot, \omega)$ berechnet (in einer Tiefe d von x) aus erreichbar ist. Das Ziel ist es nun einen Parameter-Vektor ω zu finden, sodass $\tilde{J}_d(\cdot, \omega)$ eine gute Näherung zum Optimum J^* ist. Dazu wird für jede Stellung in einer Partie $x_1, x_2, ..., x_N$ eine temporale Differenz definiert

$$d_t := \tilde{J}_d(x_{t+1}, \omega) - \tilde{J}_d(x_t, \omega)$$

und nun noch das aktualisieren des Vektors ω

$$\omega := \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, \omega)[\sum_{j=t}^{N-1} \lambda^{j-t} d_t] \tag{3}$$

Der TDLeaf(λ)-Algorithmus arbeitet nun wie folgt: Sei $\tilde{J}(\cdot, \omega)$ eine Klasse von Evaluierungsfunktionen, parametrisiert mit $\omega \in \mathbb{R}^k$. Seien weiterhin $x_1, ..., x_N$ N Stellungen, die während einer Partie betrachtet werden und $r(x_N)$ der Reward am Spielende. Als Konvention gelte: $tildeJ(x_N, w) := r(x_N)$.

(1) Für jeden Zustand x_i, berechne $\tilde{J}_d(x_i, \omega)$ unter Anwendung der MinMax-Suche bis zur Tiefe d von x_i unter Verwendung von $\tilde{J}(\cdot, \omega)$, welches die Blätter evaluiert. Beachte, dass d von Stellung zu Stellung variieren kann.

(2) Sei x_i^l das Blatt, welches ausgehend vom Knoten x_i durch die Hauptvariante erreicht wird. Sollte es mehrere Knoten geben, so wähle einen per Zufall. Beachte: $\tilde{J}_d(x_i, \omega) = \tilde{J}(x_i^l, \omega)$

(3) For t = 1 to N -1 berechne die Temporale Differenzen:
$d_t := \tilde{J}(x_{t+1}^l, \omega) - \tilde{J}(x_t^l, \omega)$

(4) Update w anhand der TD-Leaf(λ)-Formel:
$\omega := \omega + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t^l, \omega)[\sum_{j=t}^{N-1} \lambda^{j-t} d_t]$

5 KnightCap

KnightCap verbindet Temporale Differenz (TD(λ)-Algorithmus) mit der Spielbaumsuchtechnologie. Dabei wird die Bewertungsfunktion mit TDLeaf(λ) in Kombination mit dem MiniMax-Algorithmus erlernt. TDLeaf ist identisch mit TD, der Unterschied liegt lediglich darin, dass nicht auf den sich ergebenen Positionen (einer Partie) gearbeitet wird, sondern auf den Blättern des Suchbaumes (LeafNodes).

Die lineare Stellungsbewertung von KnightCap wurde nun, durch Spielen auf einigen Internetservern (z.B. FICS oder ICC), mit TDLeaf(λ) trainiert. Der Haupterfolg bestand darin, dass das Programm mit einer Bewertungsfunktion startete, deren Koeffizienten auf 0 gesetzt waren (lediglich die Materialwerte waren vorgegeben:1 für Pawn, 4 für Knight, 4 für Bishop,6 für Rookund 12 für Queen). TDLeaf wurde dazu genutzt, um durch das Internetspiel diese Koeffizienten zu aktualisieren. Im Anschluss an viele weitere Experimente wurde KnightCap mit einer verbesserten „Standardausrüstung“ versehen. Es wurde bespielsweise ein Eröffnungsbuch eingebaut. Beeindruckend ist, dass Knightcap mit einem Startrating von 1650 in nur 3 Tagen (308 Spiele) eine Stärke von 2150 erreichte. Er konnte oft die Weltmeister im Blitzschach besiegen. Baxer, Tridgell und Weaver haben auch die TD(λ) mit über 300 Spielen versucht, die Stärke erhöhte sich noch 200 Punkten. Das war eine signifikante Verbesserung, aber viel langsamer als der TD(λ) Algorithmus. Die Gründe waren:

(1) Weil alle Gewichte zu null initialisiert waren, konnten die Positionen durch kleine Änderungen relativ große verändert werden. Das heißt, nach wenigen Spielen erreichten die Programme bereits ein super Niveau.
(2) KnightCap konnte sowohl positive als auch negative Rückmeldung von den Spielen bekommen.
(3) KnightCap konnte nicht von selbst lernen. usw...

Um diese Gründe zu beweisen, hatten sie noch andere Experimente gemacht, eins davon heißt Self-Play. Baxter, Tridgell und Weaver gaben an, dass das Spiel „gegen sich selbst“ sehr viel langsamer und schlechter konvergiert, so hat z.B. eine KnightCap-Version, die in 600 Spielen gegen sich selbst gespielt hat, im Vergleichskampf gegen eine weitere Version, die 100 Spiele gegen Menschen gespielt hat, nur 11% der Spiele gewonnen. Das hat sicherlich auch damit zu tun, dass der Spielerpool auf dem Internetserver wesentlich reicher ist und ein Schachprogramm ohne zufällige Zugwahl-Parameter sich doch sehr deterministisch verhält (und damit wenig dazu lernt).

Ein weiterer Ansatz könnte sein, dass das Programm noch in Abhängigkeit der Stellung auf die Berechnung einiger Bewertungsfaktoren verzichtet, da die Berechnung beispielsweise teuer aber für die Evaluation nicht relevant sind.

6 Chinook und TDLeaf

TD-Gammon lernt durch „self-Play“ und TDLeaf sammelt die Erfahrung durch das Spiel gegen Menschen im Internet. Obwohl KnightCap mit TD Leaf Algorithmus einen großen Erfolg in nur drei Tagen hat, viele Entwickler zweifeln stets, dass TD Learning die Fähigkeit hat, ein Spielprogramm mit hohe Leistungen zu bieten. Sie haben drei Begründungen: a)KnightCap war trainiert mit Blitzschach, niemand kann die gleiche Situation mit allgemeinen Schachspielen garantieren. b)Die Stärke des KnightCaps steigt nicht mehr, bevor die Spiele die high-Performance erreichen. c)Niemand hat die von TD-Lerning trainierte Gewichte mit durch einer Experten eingestellten Gewichte verglichen.

Wie kann man sicherstellen, ob sich die von TD-gelernten Gewichte für das starke Spielen

(auch für Weltmeisterschaften) eignen? Im nächsten Abschnitt betrachten wir die Arbeit von Jonathan Schaeffer, Markian Hylnka und Vili Jussila. Sie haben versucht, die Gewichte für *Chinook* durch TDLeaf learning abzustimmen. *Chinook* ist das Weltmeist von Man-Maschinen Checkers Kampf. Seine Gewichte waren schon über fünf Jahre manuell abgestimmt. Sie waren umfassend getestet, sowohl in die „self play“ Schachspiele als auch in aberhunderte Spielen gegen Männer. Wenn ein positives Ergebnis aus Schaefers Experiment angezeigt wurde, dann kann man sagen, dass die vom TD-learning abgestimmten Gewichte auch für die High-Performance Spiele genügten.

Chinooks Evaluierungsfunktion ist eine linear Kombination von 23 wissen-basierten Merkmalen auf 4 Spielphasen. Zwei Merkmale (Werte auf ein Checker und König) können nicht modifiziert werden. Daraus ergeben sich insgesamt 84 Parameter, die während des Spieles abzustellen sind. Chinook unterstützt nur die Funktionen mit Integer, deswegen müssen die von TD Learning evaluierte Positionen abschließend zu Integer konvertiert werden.

Chinook und TD Learning arbeiten selbstständig und tauschen die Informationen mittels Text-Dateien. Solche Dateien enthalten die Informationen auf die Suchtiefe, die Sequenz von Stellung, die Rundesanzahl und die Gewichte von beiden Seiten. Nach jedem Spiel generiert Chinook eine Datei, die aus dem Resultat des Spieles und aus den Bewertungen jeder Gewichtskomponente besteht. TD Learning adaptiert die Gewichte, damit das neue Spiel immer mit abgestimmten Gewichten anfangen kann. Die Trainingsroutine war wie folgt:

(1) Chinook hat mit zwei Gewichtsdateien gegeneinander gespielt.
(2) TD Lerning modifiziert eine oder beide Gewichtsdateien nach den Zustand des Spiels.
(3) Diese Prozedur wird immer wiederholt, bis das Learning zu dem Plateau läuft.

Im diesen Prozess speichert TDL auch die Informationen über den Lernprozeß, wie z.B das Resultat von allen gespielten Partien und die Gewichtswerte nach jeder Partie, weil nicht alle Gewichte jeder Runde adaptiert werden müssen. TD-Leaf Algorithmus wird hier zur Operation auf die Zugpaaren verwendet. Die Gewichte auf Zug i wird nach der Bewertung von Zug i+1 adaptiert. Ein Update passiert nicht, wenn nur ein Zug legel ist. Um die gleichen Züge zu vermeiden, wird ein Eröffnungsbuch aufgebaut, in dem 144 drei-schichtige Anfangsstellungen gespeichert sind. Basiert auf den Erfahrungen von KnightCap, setzen die Forscher die Lernrate α=0.01 und TD-Parameter λ gegen 0,95. Aber welche Werte sind besser, bleibt immer eine Frage offen.

Zwei Annäherungen werden noch getestet. Beide, deren Gewichte wie in KnightCap auf null gesetzt sind, spielen gegen den Turnierversion von Chinook. Die erste Version ist das Lehrer-Lernen (teacher learning) und die zweite Version ist das Gegen-Selbst Lernen(self-play learning). Die Ziele sind zur Bestimmung, wie effizient der TD-Algorithmus mit high-Performance Trainer ist und ob durch Spiel gegen selbst das high-Performance Niveau erreicht wurde. In Lehrer Lernen hat Schaeffer 786 Spiele mit den 13-schichten Suchprozeduren getestet. Das Resultat gibt an, dass die Leistung der TD abgestimmten Gewichte ähnlich wie die besten der hand-abgestimmten Ergebnis sind. In self-play Lernen haben die Schwarzen 50.2% zu 48.3% gegen die Weißen gewonnen. Das Resultat

zeigt noch, dass kein Lehrer nötig war, um die TD auf Weltklasse-Niveau zu treiben. Das abschließende Resultat ist, dass die Lösungen zwischen human-getuned und TD-getuned ungefähr gleichwertig sind.

7 Zusammenfassung und Ausblick

Die Evaluierungsfunktion besteht aus zwei Teilen: die Funktionseinheiten und die Gewichte auf den Funktionen. In dieser Seminararbeit wurde gezeigt, das TD Learning eine effektive Lösung auf Suchverfahren der Gewichtung bietet. Die durch TD Learning trainierte Gewichte sind in Konkurrenz mit den durch Hand-ausgebildeten Gewichten eines high-Performance Spielprogramms. Ansonsten, haben wir auch TD und TD-Leaf Algorithmus verglichen und den Unterschied diskutiert. TD(λ) konvergiert zur linear Evaluierungsfunktion. Eine interessante Richtung ist zu beweisen, ob der TD-Leaf Algorithmus ähnliche Eingenschaft hat.

Literatur

[Baxter, 2000] J.Baxter, A.Tridgell and L.Weaver. Learning to play chess using temporal differences. *Machine Learning*, 40(3):243-263, 2000

[Schaeffer, 1997] J.Schaffer. *One Jump Ahead: Challenging Human Supremacy in Checkers.* Springer Verlag, 1997

[Baxter, 1997] J.Baxter, A.Tridgell, and L.Weaver. *KnightCap: A chess program that learns by combining TD(λ) with minimax search.*

[Tesauro,1995] G.Tesauro. Temporal difference learning and TD-Gammon. *CACM*, 38(3):58-68, 1995

[Samuel, 1959] A.Samuel. Some studies in machine learning using the game of checkers. IBM J. Of Research and Development 3, 210/229, 1959