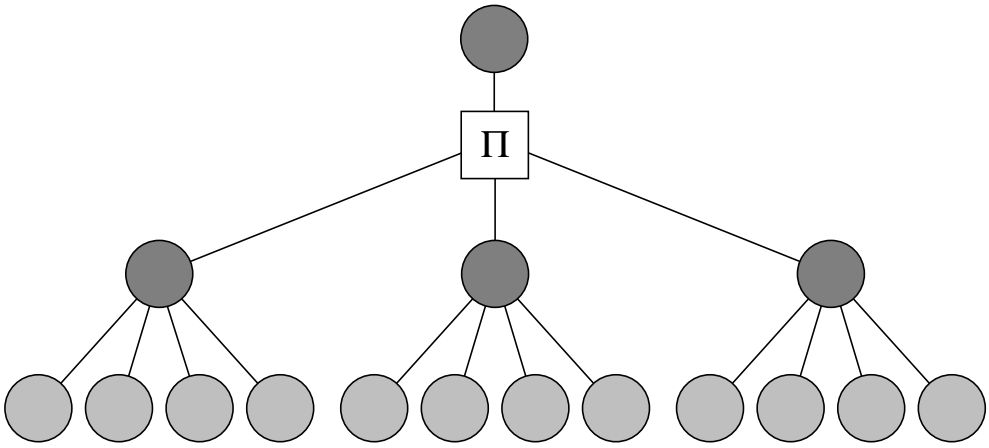
Schlüsselaustausch mit neuronalen Netzen



Diplomarbeit

vorgelegt von

Andreas Ruttor

aus Würzburg

Institut für Theoretische Physik
Bayerische Julius-Maximilians-Universität
Würzburg

Juli 2003

Inhaltsverzeichnis

Kapitel 1

Einleitung

Künstliche neuronale Netze wurden zunächst entworfen, um das Verhalten von Nervenzellen in einem mathematischen Modell analysieren und simulieren zu können. Bald zeigte sich auch, dass verschiedene komplexe Probleme der Datenverarbeitung mittels neuronaler Netze gelöst werden können. Dies gilt insbesondere für Situationen, bei denen zur Zeit des Programmentwurfs nur wenig Informationen zur Verfügung stehen. Verschiedene Methoden und Anwendungen sind beispielsweise in [1] zu finden.

Im Gegensatz zu herkömmlichen Algorithmen haben künstliche neuronale Netze die Fähigkeit aus Beispielen zu lernen und diese zu verallgemeinern. Die in dieser Arbeit betrachteten Feedforward-Netze ordnen jedem Eingabedatensatz $\vec{x}$ eine Ausgabe τ zu. Durch den Lernprozess sollen die Gewichte, die internen Parameter der Abbildung $\tau(\vec{x})$, so angepasst werden, dass ein vorgegebener Satz von Beispielen möglichst gut reproduziert wird.

Zu diesem Zweck können im Wesentlichen zwei Verfahren verwendet werden: Beim Batch-Lernen werden alle Beispiele gleichzeitig verwendet, um die optimalen Gewichte für das neuronale Netz zu ermitteln. Deshalb können auf diese Weise nur statische Zuordnungen gelernt werden. Im Gegensatz dazu wird beim Online-Lernen immer nur ein Beispiel in jedem Schritt betrachtet. Ein Lernalgorithmus gibt dann vor, wie unter Verwendung der aktuellen Eingabe, der gewünschten und der tatsächlichen Ausgabe die Gewichte anzupassen sind. In diesem Fall ist es möglich, dass sich die Abbildung, die die Beispieldaten erzeugt, mit der Zeit ändert. Eine solche dynamische Zuordnung zwischen Eingabe und richtiger Ausgabe kann auch durch ein weiteres neuronales Netz gegeben sein, dessen Gewichte sich ebenfalls verändern.

Bei der Untersuchung von miteinander wechselwirkenden Feedforward-Netzen wurde festgestellt, dass durch gegenseitiges Lernen eine Synchronisation der Gewichte zweier neuronaler Netze stattfinden kann [2]. Dies ermöglicht neue Anwendungen, insbesondere im Bereich der Kryptographie. Denn nach einer solchen Synchronisation stehen in beiden neuronalen Netzen die gleichen Gewichte zur Verfügung, obwohl diese niemals explizit in den Eingaben oder Ausgaben der

neuronalen Netze vorkamen. Auf dieser Grundlage wurde ein kryptographisches Schlüsselaustauschprotokoll in [3] vorgeschlagen.

Diese Anwendung ist nicht mit dem einfachsten Feedforward-Netz, dem Perzeptron, zu realisieren. Denn bei diesem neuronalen Netz lässt sich der Lernvorgang bei Kenntnis von Eingabe und Ausgabe vollständig rekonstruieren, da keine verborgene Zwischenschicht vorhanden ist. Bei Mehrschichtnetzwerken dagegen ist nicht der gesamte Zustand nach außen sichtbar, so dass diese für die kryptographische Anwendung besser geeignet sind. Für die praktische Verwendung ist es außerdem erforderlich, dass eine vollständige Synchronisation innerhalb einer endlichen Zeitspanne zu erreichen ist. Das kann durch die Verwendung diskreter Eingaben und Gewichte sichergestellt werden.

In dieser Arbeit werden Zweischicht-Netzwerke vom Typ der Tree Parity Machine betrachtet, die diesen Anforderungen genügen. In einem solchen neuronalen Netz arbeitet jedes Neuron der Zwischenschicht wie das Ausgabeneuron eines Perzeptrons. Die Gesamtausgabe ist dann die Parität der Zustände dieser Neuronen. Da die Eingabeneuronen jeweils nur mit einem Zwischenschichtneuron verbunden sind (baumartige Struktur), werden vom kryptographischen Standpunkt aus unerwünschte Korrelationen zwischen den Gewichten vermieden.

In Kapitel 2 wird zunächst das Prinzip des neuronalen Schlüsselaustauschs vorgestellt. Außerdem werden verschiedene Angriffsmöglichkeiten aufgezeigt. Die Synchronisation diskreter Tree Parity Machines wird anschließend in Kapitel 3 näher untersucht. Im Hinblick auf die praktische kryptographische Anwendung sind hier vor allem Erkenntnisse über Aufwand und Sicherheit des Schlüsselaustauschprotokolls von Bedeutung. Kapitel 4 behandelt dann den Feedback-Mechanismus einer Verwirrten Tree Parity Machine. Ein solches neuronales Netz lernt in jedem Schritt das Gegenteil seiner Ausgabe. Gleichzeitig wird die Folge der Ausgaben wie beim Bit-Generator als neue Eingabe verwendet. Die Eigenschaften der so erzeugten Zeitreihe werden untersucht. Insbesondere wird überprüft, ob die Ausgaben der Verwirrten Tree Parity Machine vorhersagbar oder pseudozufällig sind. Die Auswirkungen, die dieser Rückkopplungsmechanismus bei der Synchronisation hat, werden anschließend in Kapitel 5 betrachtet. Dabei wird auch ermittelt, ob sich so die Sicherheit des neuronalen Schlüsselaustauschprotokolls verbessern lässt.

Kapitel 2

Neuronaler Schlüsselaustausch

2.1 Motivation

A (Sender) will B (Empfänger) eine geheime Nachricht (z. B. per e-Mail) zukommen lassen. Er weiß aber, dass die Übertragung über viele Zwischenstationen erfolgt, an denen sein Gegner E die Nachricht problemlos mitlesen kann. Um dies zu verhindern, entschließt sich A seine Nachricht zu verschlüsseln. Wenn das verwendete Verfahren hinreichend sicher ist, bleibt der Inhalt vor dem Angreifer verborgen, solange dieser den von A verwendeten Schlüssel nicht kennt. Dasselbe gilt aber auch für B. Also muss A einen Weg finden, um mit seinem Kommunikationspartner einen Schlüssel zu vereinbaren, ohne diesen gleichzeitig dem Angreifer zu verraten.

Für dieses Problem des Schlüsselaustauschs hat man in der Kryptographie verschiedene Lösungen gefunden [4]:

- A und B verwenden für die Übertragung des Schlüssels einen anderen, sicheren Kanal. Dies ist aber oft mit erheblichem Aufwand verbunden und in manchen Situationen überhaupt nicht möglich.

- Beim Einsatz eines asymmetrischen Verschlüsselungsverfahrens kann B seinen öffentlichen Schlüssel an A schicken, ohne dass E damit die Nachricht entschlüsseln könnte. Da die Erzeugung und Verteilung dieser Schlüssel mit einem gewissen Aufwand verbunden ist, werden sie über einen längeren Zeitraum genutzt. Falls es nun E gelingt, den geheimen Schlüssel von B zu ermitteln, kennt er wahrscheinlich den Inhalt vieler Nachrichten.

- Für jede Nachricht generieren A und B einen gemeinsamen Schlüssel mittels eines Schlüsselaustauschprotokolls. Dieser wird dann auch nur für die Dauer der Nachrichtenübertragung benötigt. Eine langfristige Speicherung ist somit nicht erforderlich.

Gerade die Nutzung von Schlüsselaustauschprotokollen hat signifikante Vorteile gegenüber den beiden anderen Lösungen. Es wird weder ein sicherer Nachrichten-

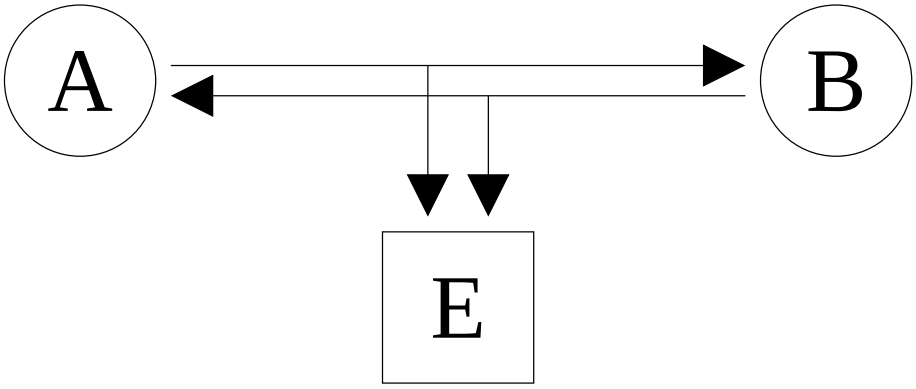


Abbildung 2.1: Situation beim Schlüsselaustausch

kanal noch eine Möglichkeit zum Aufbewahren von geheimen Schlüsseln benötigt. Deshalb können diese Methoden auch verwendet werden, um automatisiert verschlüsselte Nachrichten zwischen Computerprogrammen zu versenden.

In der Kryptographie sind einige Schlüsselaustauschprotokolle bekannt, die als sicher gelten. Das Vertrauen in diese Verfahren kommt daher, dass es bisher nicht gelungen ist, effektive Angriffsmethoden zu finden. Gleichwohl kann nicht ausgeschlossen werden, dass dies auch in Zukunft so bleibt. Somit ist es auf jeden Fall sinnvoll, nach neuen Verfahren zu suchen und ihre Sicherheit zu analysieren.

Bei der Betrachtung des in [3] vorgeschlagenen Schlüsselaustauschprotokolls müssen einige grundsätzliche Annahmen gemacht werden, um auch den Vergleich des Sicherheitsniveaus mit anderen Verfahren zu ermöglichen:

- Der Angreifer E kann alle zwischen A und B ausgetauschten Nachrichten mitlesen. Die Sicherheit der Methode hängt somit nicht vom verwendeten Übertragungskanal ab.

- E hat aber nicht die Möglichkeit, die übertragenen Nachrichten abzuändern. Um dies auszuschließen, müssen andere geeignete kryptographische oder technische Lösungen eingesetzt werden.

- Die genaue Arbeitsweise des Schlüsselaustauschs ist öffentlich bekannt. Die Geheimhaltung des Algorithmus verhindert nur, dass die Methode kryptoanalytisch untersucht wird und Schwachstellen aufgedeckt werden. Ein Angreifer kann diese trotzdem finden, so dass die Sicherheit nur scheinbar verbessert würde [4].

Basierend auf dieser Grundlage sollen nun das Prinzip des neuronalen Schlüsselaustauschs [3] vorgestellt und mögliche Angriffe beschrieben werden.

2.2 Prinzip

Die Partner A und B nutzen jeweils eine Tree Parity Machine für den Schlüsselaustausch. Die Struktur dieses neuronalen Netzes ist in Abbildung 2.2 dargestellt. Es handelt sich hierbei um ein Feedforward-Netz mit zwei Schichten. Die Zwischenschicht enthält K Neuronen (versteckte Einheiten), die jeweils wie das Ausgabeneuron eines Perzeptrons funktionieren. Zusammen mit je N Eingabeelementen x_{ij} und N Gewichten w_{ij} bilden diese Neuronen K voneinander getrennte Teilnetze, deren Ausgabe mit σ_i bezeichnet wird. Hierbei bezeichnet der Index i jeweils die versteckte Einheit ($i \in \{1, \ldots, K\}$), während j die Eingaben und Gewichte in einem Teilnetz durchnummeriert ($j \in \{1, \ldots, N\}$).

Die Eingaben sind binär, so dass die x_{ij} nur die Werte -1 oder $+1$ annehmen können. Als Gewichte werden ganze Zahlen im Bereich von $-L$ bis $+L$ eingesetzt. Wie bereits erwähnt, stellt dieser diskrete Aufbau sicher, dass eine vollständige Synchronisation in endlicher Zeit erfolgt. Die drei Parameter K, L und N werden vorher vereinbart und sind allen Beteiligten bekannt.

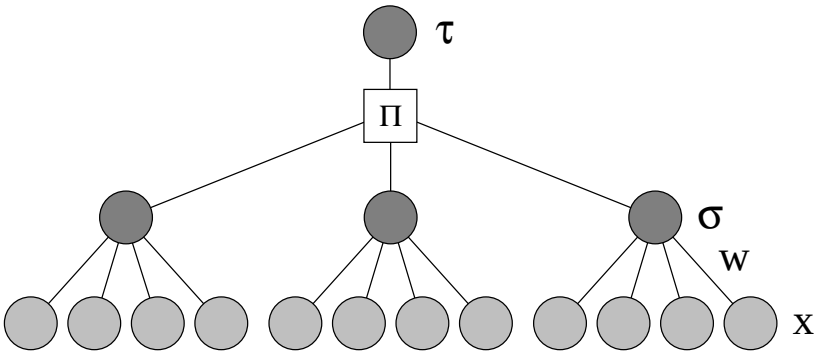


Abbildung 2.2: Struktur einer Tree Parity Machine

A und B starten mit zufällig festgelegten Gewichten. Keiner der beiden kennt den von seinem Partner gewählten Anfangszustand. Ebenso weiß der Angreifer E nicht, welche Werte die $2\,KN$ Gewichte in beiden neuronalen Netzen ursprünglich haben.

Für jeden Synchronisationsschritt wird nun eine neue zufällige Eingabe ermittelt. A und B berechnen dazu dann die Ausgabe ihres neuronalen Netzes auf folgende Weise: Zunächst wird der Zustand σ_i der Neuronen in der Zwischenschicht bestimmt. Dieser ist wie beim Perzeptron das Vorzeichen des Skalarprodukts aus Gewichts- und Eingabevektor:

$$\sigma_i = \operatorname{sign}\left(\sum_{j=1}^{N} w_{ij} x_{ij}\right) . \tag{2.1}$$

Falls das Skalarprodukt Null ist, wird σ_i gleich -1 gewählt. Die Gesamtausgabe

τ der Tree Parity Machine ergibt sich nun als Produkt bzw. Parität der Einzelausgaben:

$$\tau = \prod_{i=1}^{K} \sigma_i \,. \tag{2.2}$$

Anschließend teilen sich die beiden Partner gegenseitig die Ausgabe ihres neuronalen Netzes mit. Hiermit ist der Synchronisationsschritt beendet, wenn $\tau^A \neq \tau^B$ ist. Falls aber $\tau^A = \tau^B$ gilt, werden die Gewichte in den neuronalen Netzen angepasst. Zu diesem Zweck können verschiedene Lernregeln verwendet werden, die alle letztlich zur Synchronisation führen:

- Bei Verwendung der *Hebbschen Lernregel* lernt jedes neuronale Netz die gemeinsame Ausgabe:

$$w_{ij}^{+} = g\left(w_{ij} + \tau x_{ij} \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)\right) \,. \tag{2.3}$$

- Die vom Verwirrten Bit-Generator [5] her bekannte *Anti-Hebb-Lernregel* bewirkt, dass immer das Gegenteil der aktuellen Ausgabe gelernt wird [6]:

$$w_{ij}^{+} = g\left(w_{ij} - \tau x_{ij} \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)\right) \,. \tag{2.4}$$

- Bei der in [7] verwendeten *Random-Walk-Lernregel* lernt jede Tree Parity Machine die Ausgabe +1, wenn $\tau^A = \tau^B$ ist:

$$w_{ij}^{+} = g\left(w_{ij} + x_{ij} \Theta(\sigma_i \tau) \Theta(\tau^A \tau^B)\right) \,. \tag{2.5}$$

In jedem Fall werden nur Gewichte angepasst, die zu einer versteckten Einheit mit $\sigma_i = \tau$ gehören. Auf diese Weise ist weder dem anderen Partner noch einem Angreifer bekannt, ob ein Neuron der Zwischenschicht am Lernen teilnimmt oder nicht.

Auch ist zu beachten, dass der Wertebereich der w_{ij} beschränkt ist. Falls die Anwendung der Lernregel zu Gewichten außerhalb dieses Bereichs führt, werden die betroffenen Komponenten auf $\pm L$ gesetzt. Dies ist Aufgabe der Funktion g:

$$g(w) = \begin{cases} \operatorname{sign}(w) L & \text{für } |w| > L \\ w & \text{sonst} \end{cases} \,. \tag{2.6}$$

Mit dem Anwenden der Lernregel ist der jeweilige Synchronisationsschritt abgeschlossen. Dieser Vorgang wird nun so lange wiederholt, bis die Gewichte in den beiden neuronalen Netzen übereinstimmen.

Dann können A und B die Werte w_{ij} zum Erzeugen gemeinsamen Schlüssels verwenden. Dazu verwenden sie sinnvollerweise eine kryptographische Hash-Funktion [4], die für ähnliche Gewichtskonfigurationen verschiedene Funktionswerte liefert. Diese Maßnahme verhindert, dass der Angreifer den Schlüssel berechnen kann, wenn er nur einen Teil der Gewichte kennt. Anschließend lässt sich jedes symmetrische Verschlüsselungsverfahren zum eigentlichen Verschlüsseln der Nachricht einsetzen.

2.3 Angriffsmethoden

In den folgenden Abschnitten werden einige Angriffsmethoden vorgestellt, mit denen E versuchen kann, den von A und B mit dem Schlüsselaustauschprotokoll generierten Schlüssel zu finden.

2.3.1 Absuchen des Schlüsselraums

Für die Gewichte einer Tree Parity Machine mit den Parametern K, L und N gibt es insgesamt $(2L+1)^{KN}$ mögliche Konfigurationen. Eine dieser Varianten entspricht dem Endzustand der neuronalen Netze nach der Synchronisation und liefert den von A und B generierten Schlüssel. Theoretisch muss E also nur alle Gewichtskonfigurationen durchprobieren. Allerdings gibt es für $K = 3$, $L = 3$ und $N = 100$ bereits $3 \cdot 10^{253}$ Möglichkeiten, so dass diese Suche mit heutiger Technik nicht in vernünftiger Zeit zu bewältigen ist. Dieser Angriff beeinträchtigt die Sicherheit des neuronalen Schlüsselaustauschs also nicht, solange es nicht gelingt, die Zahl der möglichen Schlüssel einzuschränken.

2.3.2 Lernen der Ausgaben

E verhält sich schon geschickter, wenn er eine weitere Tree Parity Machine mit gleicher Struktur verwendet. Er wählt wie A und B zufällige Anfangsbedingungen. Durch das Mitlesen der Nachrichten erfährt E, in welchen Synchronisationsschritten die Gewichte angepasst werden (nämlich für $\tau^A = \tau^B$). Da ihm auch die Eingabe bekannt ist, kann er bei seinem neuronalen Netz den Lernschritt mit derselben Regel wie A und B durchführen. Dabei setzt E natürlich $\tau^{A/B}$ als Ausgabe in die Lernregel ein, auch wenn τ^E nicht damit übereinstimmt.

Wenn A und B jeweils nur ein Perzeptron ($K = 1$) verwenden würden, so hätte der Angreifer damit auch in jedem Fall Erfolg. Denn hier sind alle Informationen über die neuronalen Netze bekannt, so dass das Lernen von E auf die gleiche Weise und somit genauso schnell wie die Synchronisation von A und B abläuft.

Für $K > 1$ dagegen sind jedem Beteiligten nur für das eigene neuronale Netz die Zustände σ_i bekannt. Zu einer Ausgabe τ gibt es insgesamt 2^{K-1} mögliche interne Zustände. So ist es möglich, dass A, B und E Lernschritte in unterschiedlicher Weise durchführen.

Dadurch, dass die Ausgabe τ die Parität der σ_i ist, können A und B eine ungerade Anzahl an abweichenden Zuständen an $\tau^A \neq \tau^B$ erkennen. Sie vermeiden unkoordiniertes Lernen, indem sie in dieser Situation die Gewichte unverändert lassen. E aber muss genau dann einen Lernschritt durchführen, wenn A und B dies tun. Dies ist auch der Fall, wenn er wegen $\tau^E \neq \tau^A = \tau^B$ weiß, dass mindestens ein σ_i falsch ist. Unkoordinierte Lernschritte werden somit für den Angreifer häufiger als für die beiden Kommunikationspartner vorkommen [8]. Deshalb läuft das Lernen von E im Mittel langsamer ab als die Synchronisation von A und B.

Allerdings kann der Angreifer bei dieser Methode auch mehr als eine Tree Parity Machine einsetzen. Durch die verschiedenen Anfangsbedingungen wird die Wahrscheinlichkeit größer, dass eines der neuronalen Netze rechtzeitig die vollständige Synchronisation mit A und B erreicht.

2.3.3 Systematisches Probieren

Das Problem beim Lernen der Ausgaben ist, dass E die Zustände der Zwischenschichtneuronen von A und B nicht kennt. Stattdessen kann er aber in jedem Synchronisationsschritt alle Konfigurationen der σ_i berücksichtigen. Dazu startet E mit einem neuronalen Netz und zufällig gewählten Gewichten. In einem Lernschritt (mit $\tau^A = \tau^B$) wird jedes Netz durch 2^{K-1} Kopien mit den verschiedenen möglichen Zuständen der versteckten Einheiten ersetzt. Erst danach erfolgt die Anwendung der Lernregel.

Auf diese Weise hat der Angreifer am Ende der Synchronisation nach t Schritten $2^{(K-1)t}$ neuronale Netze, von denen eines genau dieselbe Folge von internen Zuständen durchlaufen hat wie die Tree Parity Machine von A. Dieses Netz hat deshalb die gleichen Lernschritte wie die Kommunikationspartner durchgeführt, so dass nun seine Gewichte mit hoher Wahrscheinlichkeit mit denen von A übereinstimmen.

Die typische Synchronisationszeit für $K = 3$, $L = 3$ und $N = 100$ beträgt allerdings ungefähr 400 Schritte. Somit muss E für diese Parameter insgesamt $7{\cdot}10^{240}$ neuronale Netze untersuchen, um den Schlüssel mit der hier beschriebenen Methode zu finden. Das ist besser als das Absuchen des gesamten Schlüsselraums. Dennoch ist auch diese Angriffsmethode zeitlich nicht realisierbar.

2.3.4 Genetischer Angriff

Eine Weiterentwicklung des vorhergehenden Verfahrens ist der genetische Angriff [8]. Um den Aufwand auf ein erträgliches Maß zu senken, soll jetzt die Zahl der zu betrachtenden Gewichtskonfigurationen begrenzt werden. Dazu wird ein genetischer Algorithmus eingesetzt, der bereits während der Synchronisation den Erfolg einzelner neuronaler Netze berücksichtigt.

Auch hier startet E mit einer Tree Parity Machine mit zufällig gewählten Gewichten. In jedem Schritt wird dann eine der folgenden Aktionen durchgeführt:

- Falls $\tau^A \neq \tau^B$ gilt, finden keine Änderungen statt.
- Falls $\tau^A = \tau^B$ ist und die Zahl der neuronalen Netze kleiner als eine Schranke M ist, wird jede Tree Parity Machine durch 2^{K-1} Kopien mit den möglichen internen Zuständen ersetzt. Danach wird die Lernregel auf alle neuronalen Netze angewandt.

- Falls $\tau^A = \tau^B$ ist und die Zahl der neuronalen Netze die Schranke M überschreitet, löscht E weniger erfolgreiche Netze. Solche kann er beispielsweise daran erkennen, dass sie bisher in mehr als der Hälfte der Lernschritte falsche Ausgaben ($\tau^A = \tau^B \neq \tau^E$) geliefert haben. Die anderen Tree Parity Machines werden ohne Modifikation der σ_i gemäß der Lernregel trainiert.

Der Vorteil dieses Angriffs ist, dass E maximal $2M$ neuronale Netzwerke verwenden muss. Ebenso ist die Zahl der nach Abschluss der Synchronisation zu untersuchenden Gewichtskonfigurationen nicht zu groß. Allerdings hängt die Erfolgswahrscheinlichkeit des Angreifers von der eingesetzten Löschstrategie ab.

2.3.5 Geometrischer Angriff

Diese in [8] vorgestellte Methode basiert auf geometrischen Betrachtungen. Dazu werden die Eingaben für je eine versteckte Einheit zu einem Vektor $\vec{x}_i = (x_{i1}, \ldots, x_{iN})^T$ in einem N-dimensionalen Vektorraum angesehen. Die $\vec{x}_i$ definieren als Normalenvektoren K in jedem Schritt zufällig gewählte Hyperebenen X_i durch den Ursprung des Vektorraums. In diesem Bild wird die Gewichtskonfiguration eines Zwischenschichtneurons durch einen Punkt W_i mit dem Ortsvektor $\vec{w}_i = (w_{i1}, \ldots, w_{iN})^T$ dargestellt. Der Wert von σ_i gibt dann einfach an, auf welcher Seite von X_i der Punkt W_i liegt.

Der Angreifer kann nach einigen Synchronisationsschritten annehmen, dass die Punkte W_i^A, W_i^B und W_i^E nicht allzu weit voneinander entfernt liegen. Wenn der Fall $\tau^A = \tau^B \neq \tau^E$ auftritt, kann er nun folgende Überlegungen anstellen:

- Es ist am wahrscheinlichsten, dass $\sigma_i^E \neq \sigma_i^A$ lediglich für ein i gilt. Zur Korrektur muss dann nur der richtige Index i ermittelt werden.
- Für $\sigma_i^E \neq \sigma_i^A$ liegen W_i^A und W_i^E auf verschiedenen Seiten von X_i. Dann aber sollte der Abstand zwischen W_i^E und X_i klein sein.
- Für $\sigma_i^E = \sigma_i^A$ dagegen ist es eher unwahrscheinlich W_i^E in der Nähe von X_i zu finden, weil alle Punkte im gleichen Halbraum liegen.

Der Abstand $d(W_i^E, X_i)$ zwischen W_i^E und X_i lässt sich leicht berechnen:

$$d(W_i^E, X_i) = \frac{|\vec{x}_i \cdot \vec{w}_i^E|}{|\vec{x}_i|} = \frac{1}{\sqrt{N}} \left| \sum_{j=1}^{N} x_{ij} w_{ij}^E \right| . \tag{2.7}$$

Aufgrund dieser Überlegungen wird der Angreifer vermuten, dass der falsche Wert von τ^E durch das i-te Neuron der Zwischenschicht mit minimalem Abstand $d(W_i^E, X_i)$ verursacht wird.

Auch für den geometrischen Angriff startet E mit zufällig gewählten Anfangsbedingungen. Wie schon beim einfachen Lernen der Ausgaben können ein oder

auch mehrere neuronale Netze verwendet werden. In jedem Lernschritt berechnet der Angreifer nun die Ausgabe τ^E seiner Tree Parity Machine. Abhängig vom Ergebnis führt der Angreifer dann eine der folgenden Aktionen durch:

- Für $\tau^A \neq \tau^B$ erfolgt keine Änderung der Gewichte.
- Falls $\tau^A = \tau^B = \tau^E$ gilt, verwendet E dieselbe Lernregel wie A und B.
- Gilt aber $\tau^A = \tau^B \neq \tau^E$, so sucht E den Index i mit minimalem Abstand $d(W_i^E, X_i)$. Er ändert dann das Vorzeichen von σ_i^E und τ^E, bevor die normale Lernregel angewandt wird.

Der geometrische Angriff ist der effektivste der bisher bekannten Methoden [9]. Deshalb wird insbesondere die Erfolgswahrscheinlichkeit dieser Angriffsart zur Abschätzung der Sicherheit des neuronalen Schlüsselaustauschs herangezogen.

Kapitel 3

Synchronisation und Lernen beim Schlüsselaustausch

Das in Kapitel 2 beschriebene neuronale Schlüsselaustauschprotokoll beruht darauf, dass zwei Tree Parity Machines durch wechselseitiges Lernen synchronisiert werden. Dieser Vorgang besteht aus einzelnen Schritten, in denen die Gewichte w_{ij} der beteiligten neuronalen Netze angepasst werden. Dabei hängt die Änderung nur von der aktuellen Eingabe x_{ij} ab, die auch die Ausgaben σ_i und τ festlegt. Deshalb können die einzelnen Lernschritte unabhängig voneinander betrachtet werden. Die neuen, angepassten Gewichte werden dann mit w_{ij}^+ bezeichnet.

Die Wirkung eines Lernschritts auf die Synchronisation zweier Zwischenschichtneuronen lässt sich am besten beurteilen, indem die Änderung des Überlapps ρ untersucht wird. Dieser Ordnungsparameter hängt gemäß $\rho = \cos\phi$ vom Winkel ϕ zwischen den Gewichtsvektoren der beiden versteckten Einheiten ab. Ein attraktiver Lernschritt ist daran zu erkennen, dass der Abstand der Gewichte und folglich auch ϕ kleiner wird. Bei einem repulsiven Schritt dagegen vergrößert sich der Winkel zwischen den Gewichtsvektoren.

Für die Anwendung des neuronalen Schlüsselaustauschs in der Kryptographie ist vor allem die Sicherheit gegen Angriffe von großer Bedeutung. Deshalb soll hier die Erfolgswahrscheinlichkeit P_E eines Angreifers durch Simulationen ermittelt werden. Dabei zeigt sich auch, welche Methoden besonders effektiv sind, so dass man sich bei der Verbesserung der Sicherheit auf die Abwehr dieser Angriffe konzentrieren kann. Allerdings lohnt sich der Einsatz des neuronalen Schlüsselaustauschs nur, wenn der Aufwand des Verfahrens nicht zu groß ist. Deshalb ist auch eine Bestimmung der Synchronisationszeit erforderlich.

Mit Simulationen ist aber nur die Betrachtung kleiner Systeme möglich. Hier können zusätzliche finite-size-Effekte auftreten, die eventuell allgemeine Gesetzmäßigkeiten verdecken. Deshalb sind auch Untersuchungen an großen Systemen nötig. Für solche Berechnungen kann ein iteratives Verfahren [10, 11] genutzt werden, das im letzten Abschnitt dieses Kapitels erläutert wird.

3.1 Synchronisation der versteckten Einheiten

Zunächst soll die Wirkung eines Synchronisationsschritts auf die Zwischenschichtneuronen der Tree Parity Machine untersucht werden. Dazu werden folgende Ordnungsparameter [11] betrachtet:

$$Q_i^m = \frac{1}{N}\sum_{j=1}^{N}(w_{ij}^m)^2\,; \tag{3.1}$$

$$R_i^{mn} = \frac{1}{N}\sum_{j=1}^{N} w_{ij}^m w_{ij}^n\,. \tag{3.2}$$

Die Indizes m und n kennzeichnen hier, zu welchem der Beteiligten das betrachtete neuronale Netz gehört ($m, n \in \{A, B, E\}$).

Aus diesen Größen lässt sich dann leicht der Überlapp ρ_i^{mn} zweier korrespondierender Neuronen berechnen:

$$\rho_i^{mn} = \frac{R_i^{mn}}{\sqrt{Q_i^m Q_i^n}}\,. \tag{3.3}$$

An ρ_i^{mn} kann der Grad der erreichten Synchronisation abgelesen werden. Wenn die Gewichte beider Zwischenschichtneuronen völlig unkorreliert sind, so gilt $\rho = 0$. Für $\rho = 1$ dagegen stimmt die Ausgabe σ_i der Neuronen für jede mögliche Eingabe überein.

3.1.1 Random Walk der Gewichte

Die Änderung der Gewichte in einem Synchronisationsschritt hängt von der verwendeten Lernregel ab. Allerdings liegt den in Kapitel 2.2 vorgestellten Möglichkeiten der gleiche Algorithmus zugrunde. In jedem Fall werden die Eingabeelemente x_{ij} mit einem von der Lernregel festgelegten Vorfaktor $f(\tau, \sigma)$ multipliziert und anschließend zu den Gewichten w_{ij} addiert:

$$w_{ij}^+ = g(w_{ij} + x_{ij} f(\tau, \sigma_i))\,. \tag{3.4}$$

Da die x_{ij} zufällig gewählt werden, beschreibt Gleichung (3.4) einen Random Walk der Gewichte [12]. Allerdings bestimmt der Faktor $f(\tau, \sigma) \in \{-1, 0, +1\}$, welche Eingaben ausgelassen werden müssen, um eine Synchronisation zu erreichen:

$$f(\tau,\sigma) = \Theta(\sigma\tau)\,\Theta(\tau^A\tau^B)\cdot \begin{cases} \tau & \text{Hebbsche Lernregel} \\ -\tau & \text{Anti-Hebb-Lernregel} \\ 1 & \text{Random-Walk-Lernregel} \end{cases}. \tag{3.5}$$

Zusätzlich ist zu berücksichtigen, dass der Wertebereich der Gewichte auf ganze Zahlen von $-L$ bis $+L$ beschränkt ist. Deshalb bleiben die Gewichte unverändert,

deren neuer Wert w_{ij}^{+} außerhalb des zulässigen Bereichs liegen würde. Diese reflektierende Randbedingung wird durch die in Gleichung (2.6) definierte Funktion $g(w)$ umgesetzt.

Falls die Hebbsche Lernregel oder die Anti-Hebb-Lernregel gewählt wird, hängt gemäß Gleichung (3.5) die Richtung der Gewichtsänderung auch von τ ab. Dadurch ist der Random Walk eines Gewichts w_{ij} nicht mehr unabhängig vom Zustand des restlichen neuronalen Netzes. Effekte, die sich hieraus ergeben, sollen aber erst in Kapitel 3.1.5 berücksichtigt werden.

Betrachtet man das i-te Zwischenschichtneuron in den Tree Parity Machines von A und B, so tritt in jedem Synchronisationsschritt eine der folgenden Möglichkeiten auf:

- Falls die Gesamtausgaben der beiden neuronalen Netze nicht übereinstimmen ($\tau^A \neq \tau^B$), bleiben alle Gewichte unverändert.
- Wenn sowohl $\tau^A \neq \sigma_i^A$ als auch $\tau^B \neq \sigma_i^B$ gilt, hat der Lernschritt auf die i-te versteckte Einheit keine Auswirkung. Die Gewichte bleiben dort unverändert.
- Ein *attraktiver Lernschritt* findet im i-ten Zwischenschichtneuron statt, wenn $\tau^A = \tau^B = \sigma_i^A = \sigma_i^B$ gilt.
- Ein *repulsiver Lernschritt* ist durch $\tau^A = \tau^B$ und $\sigma_i^A \neq \sigma_i^B$ gekennzeichnet.

Auf die Wirkung der attraktiven und repulsiven Schritte soll im Folgenden näher eingegangen werden.

3.1.2 Berechnung des Überlapps

Der Synchronisationsgrad einzelner Gewichte w_{ij}^A und w_{ij}^B kann am besten durch ihren Abstand d_{ij} beschrieben werden:

$$d_{ij} = w_{ij}^A - w_{ij}^B \,. \tag{3.6}$$

In ähnlicher Weise kennzeichnet das Abstandsquadrat D_i der Gewichtsvektoren $\vec{w}_i^A$ und $\vec{w}_i^B$ die bereits erzielte Übereinstimmung zwischen zwei versteckten Einheiten:

$$D_i = \sum_{j=1}^{N} d_{ij}^2 \,. \tag{3.7}$$

Die vollständige Synchronisation zweier korrespondierender Zwischenschichtneuronen ist dann an $D_i = 0$ zu erkennen.

Die Abstände d_{ij} allein reichen aber nicht aus, um die Werte der Gewichte vollständig zu beschreiben. Deshalb ist es für die Berechnung des Überlapps ρ_i^{AB} sinnvoll, auch die Summen s_{ij} der Gewichte von A und B zu definieren:

$$s_{ij} = w_{ij}^A + w_{ij}^B \,. \tag{3.8}$$

Durch Angabe der d_{ij} und s_{ij} sind auch die w_{ij}^A und w_{ij}^B eindeutig bestimmt. Zur Umrechnung ist lediglich eine lineare Transformation anzuwenden.
Berechnet man nun die Ordnungsparameter aus diesen Größen, so ergibt sich:

$$Q_i^A = \frac{1}{4N}\sum_{j=1}^{N}(s_{ij}^2 + 2s_{ij}d_{ij} + d_{ij}^2)\,; \tag{3.9}$$

$$Q_i^B = \frac{1}{4N}\sum_{j=1}^{N}(s_{ij}^2 - 2s_{ij}d_{ij} + d_{ij}^2)\,; \tag{3.10}$$

$$R_i^{AB} = \frac{1}{4N}\sum_{j=1}^{N}(s_{ij}^2 - d_{ij}^2)\,. \tag{3.11}$$

Diese Beziehungen lassen sich noch vereinfachen, indem zwei weitere Größen analog zu D_i definiert werden:

$$S_i = \sum_{j=1}^{N} s_{ij}^2\,; \tag{3.12}$$

$$\gamma_i = \sum_{j=1}^{N} s_{ij}d_{ij}\,. \tag{3.13}$$

Dabei bezeichnet γ_i das Skalarprodukt der Gewichtsvektoren $\vec{w}_i^A$ und $\vec{w}_i^B$. S_i ist das Betragsquadrat der Summe $\vec{w}_i^A + \vec{w}_i^B$.
Für die Ordnungsparameter gilt dann:

$$Q_i^A = \frac{1}{4N}(S_i + 2\gamma_i + D_i)\,; \tag{3.14}$$

$$Q_i^B = \frac{1}{4N}(S_i - 2\gamma_i + D_i)\,; \tag{3.15}$$

$$R_i^{AB} = \frac{1}{4N}(S_i - D_i)\,. \tag{3.16}$$

Daraus folgt für den Überlapp:

$$\rho_i^{AB} = \frac{S_i - D_i}{\sqrt{(S_i + D_i)^2 - 4\gamma_i^2}}\,. \tag{3.17}$$

Weil beide neuronalen Netze gleich aufgebaut sind, ist die Annahme berechtigt, dass Q_i^A ungefähr gleich Q_i^B ist. Wegen $\gamma_i = N(Q_i^A - Q_i^B)$ folgt daraus aber $\gamma_i \approx 0$. Deshalb darf $4\gamma_i^2$ für die Berechnung der Ordnungsparameter meistens gegenüber $(S_i + D_i)^2$ vernachlässigt werden. Dies wird auch durch entsprechende Simulationen bestätigt, wie in Abbildung 3.1 zu sehen ist. Zu jedem Zeitpunkt

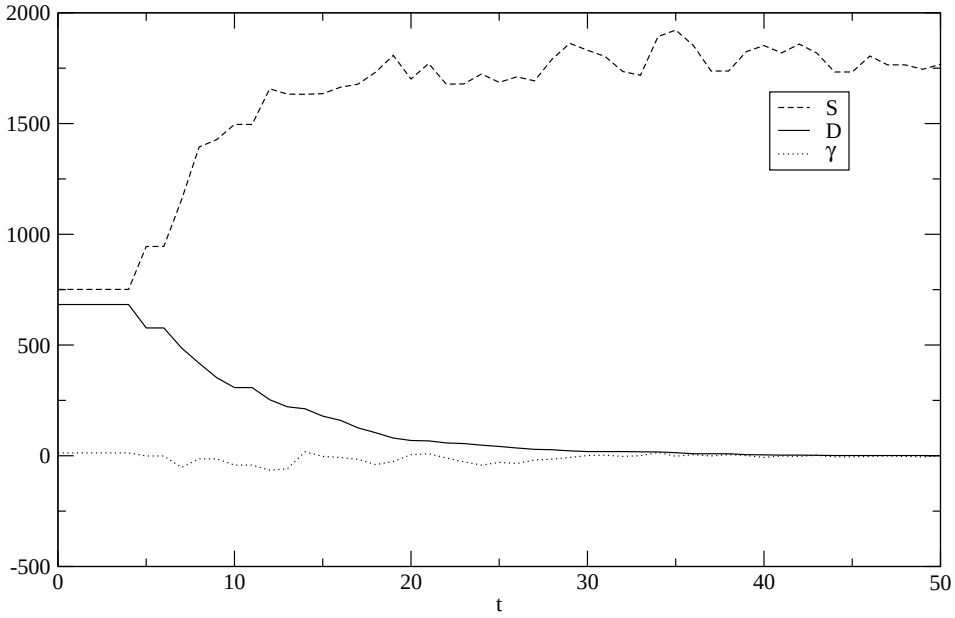


Abbildung 3.1: Synchronisationsverlauf bei einer einzelnen Simulation unter Verwendung der Hebbschen Lernregel für $K = 1$, $L = 3$ und $N = 100$.

ist die Summe $S_i + D_i$ viel größer als γ_i. Somit kann ρ_i^{AB} in guter Näherung unabhängig von γ_i berechnet werden:

$$\rho_i^{AB} \approx \frac{S_i - D_i}{S_i + D_i}\,. \tag{3.18}$$

Der Überlapp hängt hier nur noch von dem Verhältnis D_i/S_i ab. Anhand von Abbildung 3.1 erkennt man aber, dass im allgemeinen $D_i < S_i$ gilt. Änderungen der Abstände zweier Gewichte wirken sich deshalb stärker auf ρ_i^{AB} aus als Schwankungen der s_{ij}. Es kommt daher bei der Untersuchung der attraktiven und repulsiven Wirkung einzelner Lernschritte hauptsächlich auf die d_{ij} an.

3.1.3 Attraktive Schritte

Ein attraktiver Schritt erfolgt für $\tau^A = \tau^B = \sigma_i^A = \sigma_i^B$. In diesem Fall werden die w_{ij} zweier korrespondierender Zwischenschichtneuronen in gleicher Weise angepasst. Solange sich ein Gewicht in beiden neuronalen Netzen im Inneren des Wertebereichs bewegt, bleibt der Abstand konstant. Findet aber eine Reflexion eines noch nicht synchronisierten Gewichts am Rand des Wertebereichs statt, so wird der Betrag $|d_{ij}|$ des Abstands um eins verringert. Denn in diesem Fall bewegt sich ein Gewicht auf das andere zu, während Letzteres unverändert bleibt. Eine Verkleinerung der $|d_{ij}|$ führt dazu, dass auch D_i abnimmt. Das bewirkt dann gemäß Gleichung (3.18) eine Vergrößerung des Überlapps ρ_i^{AB}.

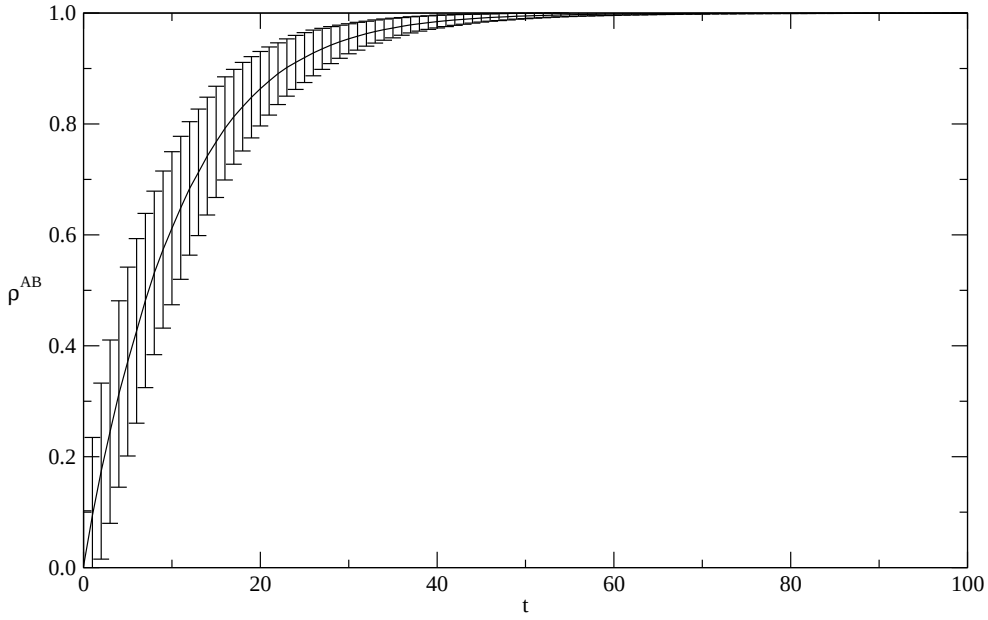


Abbildung 3.2: Verlauf des Überlapps bei Verwendung der Random-Walk-Lernregel mit den Parametern $K = 1$, $L = 3$ und $N = 100$, gemittelt über 1000 Simulationen. Die Fehlerbalken geben die Standardabweichung von ρ^{AB} an.

Die Wirkung attraktiver Schritte wird vor allem deutlich, wenn die Synchronisation zweier Perzeptrons ($K = 1$) betrachtet wird. Da keine versteckten Einheiten vorhanden sind, treten in diesem Fall repulsive Schritte nicht auf.

Abbildung 3.2 zeigt einen Verlauf des Überlapps, der nur durch attraktive Schritte hervorgerufen wird. Dabei stellt man fest, dass zunächst ein rascher Anstieg von ρ erfolgt, der aber mit zunehmender Synchronisation flacher verläuft.

Die Anzahl der möglichen Positionen für ein Paar von Gewichten ergibt sich zu $(2L+1)-|d_{ij}|$. Weil ihnen weniger Plätze zur Verfügung stehen, stoßen Gewichte-Paare mit großem $|d_{ij}|$ beim Random Walk häufiger an die Ränder des Wertebereichs. Dadurch nehmen große Abstände im Verlauf der Synchronisation schneller ab, was anfangs große Änderungen von D_i bewirkt. Später sind die $|d_{ij}|$ kleiner, so dass sich ρ langsamer ändert. Dies erklärt den in Abbildung 3.2 beobachteten Verlauf.

3.1.4 Repulsive Schritte

Bei einem repulsiven Schritt stimmen die Ausgaben σ_i der beiden korrespondierenden Neuronen in der Zwischenschicht nicht überein. Da trotzdem $\tau^A = \tau^B$ gilt, ändern sich die Gewichte in einer der beiden versteckten Einheiten, während sie in der anderen gleich bleiben. In den meisten Fällen wird dadurch die bereits vorhandene Übereinstimmung zwischen den Gewichten zerstört. Nur wenn die Randbedingung verhindert, dass sich das Gewicht des Zwischenschichtneurons

mit $\sigma_i = \tau$ bewegt, bleibt $d_{ij} = 0$ erhalten. Also ist die Wirkung eines repulsiven Schritts besonders groß, wenn bereits eine fast vollständige Synchronisation erreicht worden ist.

Für $d_{ij} \neq 0$ ist sowohl ein Anwachsen wie auch ein Abnehmen des Abstands möglich. Da die d_{ij} quadratisch in die Summe D_i eingehen, wirkt sich eine Vergrößerung des Abstands stärker aus als eine Annäherung der Gewichte. Dies lässt sich bereits bei der Betrachtung zweier Gewichte von A erkennen, die beide vor dem repulsiven Schritt denselben Abstand d von den zugehörigen Gewichten im neuronalen Netz von B hatten. Wenn sich nun nur der Abstand des einen Gewichts vergrößert, der des anderen aber verkleinert, so erhöht sich D um $\Delta D = (d+1)^2 + (d-1)^2 - 2d^2 = 2$.

Damit also D_i beim repulsiven Schritt nicht anwächst, müssten deutlich mehr Abstände $|d_{ij}|$ kleiner als größer werden. Dies ist aber bei keiner der Lernregeln zu erwarten. Deshalb wird normalerweise D_i durch diesen Schritt vergrößert. Folglich führt der repulsive Schritt gemäß Gleichung (3.18) zur Verringerung des Überlapps ρ_i^{AB}.

3.1.5 Unterschiede zwischen den Lernregeln

Die Wirkung der attraktiven und repulsiven Schritte hängt hauptsächlich davon ab, ob die Bewegung der w_{ij} koordiniert abläuft. Deshalb sind beim Verlauf des Überlapps ρ_i^{AB} auch keine qualitativen Unterschiede zu erwarten, wenn verschiedene Lernregeln verwendet werden. Somit konnte der Einfluss der Lernregeln auf den Random Walk der Gewichte bisher vernachlässigt werden.

Allerdings bleibt die ursprüngliche Gleichverteilung der Gewichte nur dann erhalten, wenn die beiden möglichen Schritte beim Random Walk mit gleicher Wahrscheinlichkeit auftreten. Dies trifft für die Hebbsche Lernregel und die Anti-Hebb-Lernregel aber nicht zu, weil die Ausgabe τ der Tree Parity Machine die Richtung der Gewichtsänderung beeinflusst. Deshalb ist zu erwarten, dass sich die Verteilung der w_{ij} ändert, wenn eine der beiden Lernregeln eingesetzt wird. Dieser Effekt sollte sich auch auf die Länge der Gewichtsvektoren auswirken.

Der in Gleichung (3.1) definierte Ordnungsparameter Q_i^m ist proportional zum Längenquadrat des Gewichtsvektors. Für seine Änderung in einem Synchronisationsschritt gilt ohne Berücksichtigung der Randbedingung $|w_{ij}| \leq L$:

$$Q_i^{m+} = \frac{1}{N}(\vec{w}_i^m + f(\tau^m, \sigma_i^m)\vec{x}_i)^2 \,. \tag{3.19}$$

Gleichung (3.19) lässt sich vereinfachen, indem das lokale Feld h_i des Zwischenschichtneurons eingesetzt wird. Diese Zufallsgröße ist bis auf einen Vorfaktor $N^{-1/2}$ gleich dem Skalarprodukt aus Gewichts- und Eingabevektor [11]:

$$h_i^m = \frac{1}{\sqrt{N}}\, \vec{w}_i^m \vec{x}_i \,. \tag{3.20}$$

Weil die beiden möglichen Eingabewerte $+1$ und -1 mit gleicher Wahrscheinlichkeit auftreten, verschwindet der Erwartungswert des lokalen Feldes:

$$\langle h_i^m \rangle = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} w_{ij}^m \langle x_{ij} \rangle = 0 \,. \tag{3.21}$$

Allerdings haben die h_i^m eine endliche Varianz:

$$\left\langle (h_i^m)^2 \right\rangle - (\langle h_i^m \rangle)^2 = \frac{1}{N} \sum_{r=1}^{N} \sum_{s=1}^{N} w_{ir}^m w_{is}^m \langle x_{ir} x_{is} \rangle = Q_i^m \,. \tag{3.22}$$

Wegen des Zentralen Grenzwertsatzes [13] sind die lokalen Felder für große N näherungsweise normalverteilte Zufallsvariablen mit Mittelwert $\mu_h = 0$ und Standardabweichung $\sigma_h = \sqrt{Q_i^m}$.

Zur Vereinfachung von Gleichung (3.19) kann auch noch verwendet werden, dass der Eingabevektor $\vec{x}_i$ immer die Länge $\sqrt{N}$ hat. Daraus folgt dann für das neue Q_i^{m+} nach einem Synchronisationsschritt:

$$Q_i^{m+} = Q_i^m + \frac{2}{\sqrt{N}} f(\tau^m, \sigma_i^m)\, h_i^m + f(\tau^m, \sigma_i^m)^2 \,. \tag{3.23}$$

Falls die Ausgaben der neuronalen Netze von A und B nicht übereinstimmen, findet wegen $f(\tau^m, \sigma_i^m) = 0$ auch keine Änderung von Q_i^m statt. Deshalb sollen im Folgenden nur Lernschritte mit $\tau^A = \tau^B = \sigma_i^m$ betrachtet werden, für die $f(\tau^m, \sigma_i^m) \neq 0$ gilt.

- Wird die *Hebbsche Lernregel* verwendet, so gilt bei einer Anpassung der Gewichte $f(\tau, \sigma) = \tau$. Wegen $\tau = \sigma_i = \mathrm{sign}(h_i)$ ist das Produkt aus h_i und Ausgabe τ dann gleich dem Betrag des lokalen Feldes. Ohne Berücksichtigung der Randbedingung ergibt sich somit:

  $$Q_i^{m+} = Q_i^m + \frac{2}{\sqrt{N}} |h_i^m| + 1 \,. \tag{3.24}$$

  Die Hebbsche Lernregel bewirkt also, dass bei jeder Änderung der Gewichte die Länge von $\vec{w}_i$ vergrößert wird. Diesem Anwachsen sind aber durch die Bedingung $|w_{ij}| \leq L$ Grenzen gesetzt, so dass dieser Effekt nur zu Beginn der Synchronisation auftritt. Danach bleibt Q_i^m nahezu konstant (siehe Abbildung 3.3).

- Die *Anti-Hebb-Lernregel* unterscheidet sich von der vorher betrachteten Hebbschen Lernregel nur durch die Richtung des Random Walks. Dadurch ändert sich auch das Vorzeichen, mit dem die Beträge der lokalen Felder in die Gleichungen für den Ordnungsparameter Q_i^m eingehen:

  $$Q_i^{m+} = Q_i^m - \frac{2}{\sqrt{N}} |h_i^m| + 1 \,. \tag{3.25}$$

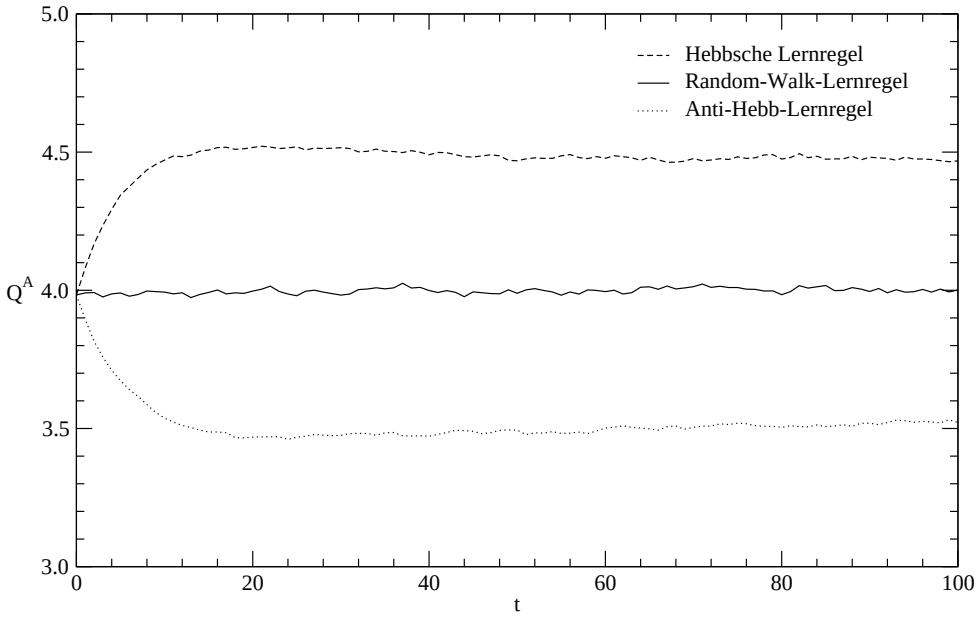


Abbildung 3.3: Verlauf von Q^A während der Synchronisation zweier neuronaler Netze mit $K = 1$, $L = 3$ und $N = 100$. Es wurde über 1000 Simulationen gemittelt.

Bei dieser Lernregel kann Q_i^m auch wieder kleiner werden. Da die Standardabweichung des lokalen Feldes gleich $\sqrt{Q_i^m}$ ist, werden insbesondere lange Gewichtsvektoren stark verkürzt. Deshalb divergieren die w_{ij} auch ohne eine Beschränkung des Wertebereichs nicht [5]. In den Simulationen beobachtet man, dass bei Verwendung der Anti-Hebb-Lernregel Q_i^m zunächst kleiner wird, um dann nahezu konstant zu bleiben. Das ist in Abbildung 3.3 deutlich zu erkennen.

- Bei der *Random-Walk-Lernregel* wird die Änderungsrichtung der Gewichte nur durch den Eingabevektor $\vec{x}_i$ festgelegt. Da die x_{ij} mit gleicher Wahrscheinlichkeit die Werte $+1$ und -1 annehmen, kann es keine Vorzugsrichtung geben. Somit strebt die Random-Walk-Lernregel eine möglichst gleichmäßige Verteilung der Gewichte an. Dieser Zustand besteht jedoch schon am Anfang der Synchronisation, weil die w_{ij} zufällig initialisiert werden. Deshalb sollte sich die Länge des Gewichtsvektors kaum verändern. Abbildung 3.3 zeigt, dass dies tatsächlich der Fall ist.

Die Gleichungen, die für verschiedene Lernregeln die Änderung der Ordnungsparameter beschreiben, unterscheiden sich nur dadurch, wie das lokale Feld h_i^m in die Berechnung eingeht. Dies zeigt bereits ein Vergleich der Formeln (3.24) und (3.25). Deshalb sollten die Unterschiede zwischen den Lernregeln verschwinden, wenn der Einfluss von h_i^m vernachlässigt werden kann. Der Parameterbereich, für

den diese Näherung zulässig ist, ergibt sich aus einer Abschätzung des Erwartungswerts $\langle|h_i^m|\rangle$:

$$\langle|h_i^m|\rangle = \sqrt{\frac{2Q_i^m}{\pi}} \leq \sqrt{\frac{2}{\pi}}L\,. \tag{3.26}$$

Hier wurde verwendet, dass h_i^m näherungsweise als normalverteilte Zufallsgröße betrachtet werden kann, wenn die Zahl N der Gewichte in einer versteckten Einheit groß ist. Außerdem gilt $Q_i^m \leq L^2$ wegen der Randbedingung $|w_{ij}^m| \leq L$. Daraus folgt für den Beitrag des lokalen Feldes in den Gleichungen (3.24) und (3.25):

$$\left\langle \frac{2}{\sqrt{N}}\,|h_i^m| \right\rangle \leq \sqrt{\frac{8}{\pi}}\frac{L}{\sqrt{N}}\,. \tag{3.27}$$

Der Einfluss von h_i^m auf die Änderung der Ordnungsparameter kann also vernachlässigt werden, wenn $L \ll \sqrt{N}$ gilt. In diesem Fall wird auch die Länge der Gewichtsvektoren unabhängig von der Wahl der Lernregel. Bereits für $L = 3$ und $N = 10\,000$ sind die Unterschiede im Verlauf von Q_i^m ziemlich klein (siehe Abbildung 3.4).

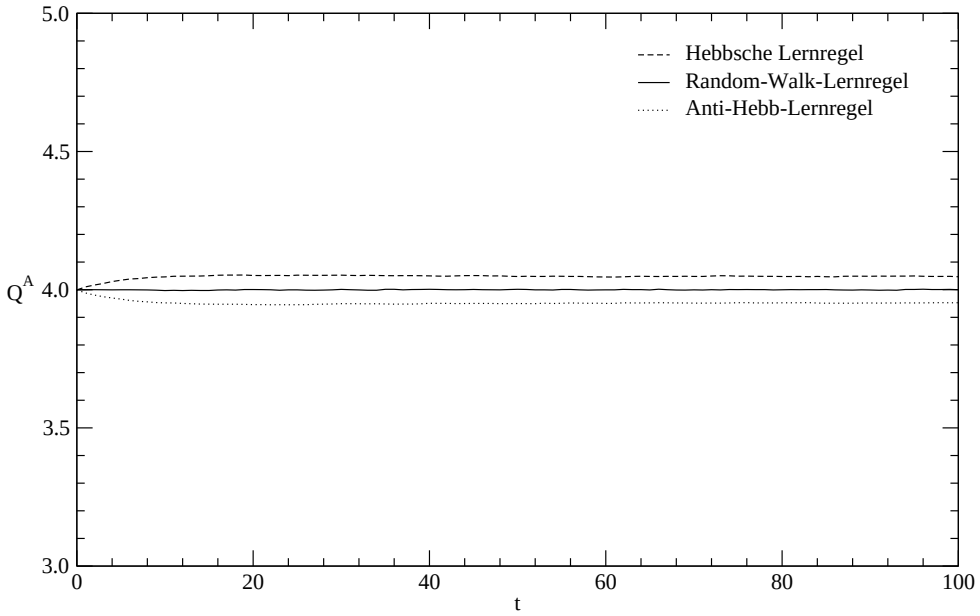


Abbildung 3.4: Verlauf von Q^A während der Synchronisation zweier neuronaler Netze mit den Parametern $K = 1$, $L = 3$ und $N = 10\,000$, gemittelt über 1000 Simulationen.

3.2 Häufigkeit repulsiver Schritte

Für die Sicherheit des neuronalen Schlüsselaustauschs ist es wichtig, dass die Kommunikationspartner A und B Vorteile gegenüber einem Angreifer E haben. Dabei spielt die Häufigkeit der attraktiven und repulsiven Schritte in den einzelnen versteckten Einheiten der neuronalen Netze eine wichtige Rolle. Deshalb verwendet man für den Schlüsselaustausch Tree Parity Machines mit $K > 1$, weil bei einfachen Perzeptrons ($K = 1$) alle Synchronisationsschritte entweder attraktiv sind oder keine Änderung der Gewichte bewirken.

Ein repulsiver Schritt in einer versteckten Einheit kann nur dann auftreten, wenn die entsprechenden Zwischenschichtneuronen der beiden betrachteten neuronalen Netze unterschiedliche Zustände σ_i aufweisen. Die Wahrscheinlichkeit ϵ_i^{mn} für eine solche Abweichung hängt vom Überlapp ρ_i^{mn} ab, der in Gleichung (3.3) definiert wurde:

$$\epsilon_i^{mn} = \frac{1}{\pi} \arccos \rho_i^{mn} . \tag{3.28}$$

ϵ_i^{mn} wird als Verallgemeinerungsfehler bezeichnet. Denn diese Größe gibt auch die Wahrscheinlichkeit an, mit der ein Perzeptron, das den Überlapp ρ zum Lehrer hat, ein neues Muster falsch klassifiziert [14].

3.2.1 Angriff durch Lernen der Ausgaben

Ein Angreifer kann, wie in Kapitel 2.3.2 beschrieben, eine weitere Tree Parity Machine trainieren. Dazu verwendet er die zwischen A und B ausgetauschten Eingaben x_{ij} und die zugehörigen Ausgaben als Beispiele.

Die Gewichte der neuronalen Netze von A und B werden genau dann angepasst, wenn $\tau^A = \tau^B$ ist. In diesem Fall müssen die w_{ij}^E auch dann verändert werden, wenn dieser Schritt wegen $\tau^E \neq \tau^A$ in mindestens einer versteckten Einheit repulsiv wirkt. Sonst würde sich nämlich der Überlapp ρ_i^{AE} für alle $i \in \{1, \ldots, K\}$ verringern. Wenn aber $\tau^A \neq \tau^B$ ist, sollte auch E keine Gewichte anpassen, weil dies ebenfalls eine repulsive Wirkung hätte.

Also hat der Zustand der Tree Parity Machine von E keinen Einfluss darauf, ob die Gewichte – wegen $\tau^A = \tau^B$ – im jeweiligen Zeitschritt angepasst werden. Deshalb gibt der Verallgemeinerungsfehler direkt die Wahrscheinlichkeit an, dass die Änderung der w_{ij}^E im i-ten Teilnetz repulsiv wirkt [8]:

$$P(\sigma_i^E \neq \sigma_i^A | \tau^A = \tau^B) = \epsilon_i^{AE} . \tag{3.29}$$

Diese Beziehung gilt aber nur, wenn der Angreifer lediglich die Ausgaben der Kommunikationspartner lernt. Die Auswirkungen verbesserter Angriffsmethoden werden in Kapitel 3.2.3 untersucht.

3.2.2 Unterschied zwischen Synchronisation und Lernen

Im Gegensatz zu E können die Kommunikationspartner A und B auswählen, in welchen Schritten die Gewichte verändert werden. Diese Möglichkeit nutzen sie, um repulsive Schritte zu vermeiden. Dazu bleiben gemäß der Lernregel die Gewichte unverändert, wenn $\tau^A \neq \tau^B$ ist. In diesem Fall gilt nämlich für mindestens eine versteckte Einheit $\sigma_i^A \neq \sigma_i^B$, so dass eine Anpassung der $w_{ij}^{A/B}$ repulsiv wirken würde. Dadurch sind die Kommunikationspartner gegenüber einem Angreifer im Vorteil, der auf den Zeitpunkt einer Änderung der Gewichte keinen Einfluss hat. Dies zeigt schon eine einfache Überlegung in [8], die hier für $K = 3$ erläutert werden soll.

Es soll die Wahrscheinlichkeit P_R berechnet werden, mit der ein Lernschritt in einer festgelegten versteckten Einheit repulsiv wirkt. Zur Vereinfachung wird dabei angenommen, dass der Verallgemeinerungsfehler für alle Zwischenschichtneuronen gleich ist ($\epsilon_i^{mn} := \epsilon$). Die Wahrscheinlichkeit P_R^E für E kann direkt aus Gleichung (3.29) abgelesen werden:

$$P_R^E = P(\sigma_i^E \neq \sigma_i^A | \tau^A = \tau^B) = \epsilon \,. \tag{3.30}$$

Bei der Berechnung von P_R^B ist zu beachten, dass τ die Parität der internen Zustände σ_i ist. Deshalb kann $\tau^A = \tau^B$ nur dann vorkommen, wenn für eine gerade Zahl der versteckten Einheiten $\sigma_i^A \neq \sigma_i^B$ gilt. Bei Tree Parity Machines, die jeweils drei Teilnetze enthalten, tritt diese Situation mit der Wahrscheinlichkeit $P(\tau^A = \tau^B) = (1-\epsilon)^3 + 3(1-\epsilon)\epsilon^2$ auf.
Daraus folgt für B:

$$P_R^B = P(\sigma_i^B \neq \sigma_i^A | \tau^A = \tau^B) = \frac{2(1-\epsilon)\epsilon^2}{(1-\epsilon)^3 + 3(1-\epsilon)\epsilon^2} \,. \tag{3.31}$$

Bei dieser Berechnung wurde angenommen, dass zwischen den Gewichten der verschiedenen Teilnetze keine Korrelationen bestehen. Dies ist durch die Verwendung getrennter Eingaben für alle versteckten Einheiten sichergestellt.

Beim Vergleich von P_R^E und P_R^B ist festzustellen, dass im neuronalen Netz des Angreifers repulsive Schritte häufiger auftreten als bei den beiden Kommunikationspartnern. Lediglich für die Spezialfälle $\rho = 0$ und $\rho = 1$ sind P_R^E und P_R^B gleich groß (siehe Abbildung 3.5).

Das gilt auch für andere Werte von K, solange $K > 1$ ist. Allerdings kann mit der Parität τ nur eine ungerade Anzahl an Abweichungen $\sigma_i^A \neq \sigma_i^B$ festgestellt werden. Deshalb ist die Parität zur Fehlererkennung weniger gut geeignet, wenn $K \gg 1$ ist. Abbildung 3.5 zeigt, dass A und B bei $K = 2$ den maximalen Vorteil gegenüber E haben. Hier funktionieren jedoch andere Angriffsmethoden sehr gut, zum Beispiel der genetische Angriff, so dass beim neuronalen Schlüsselaustausch mindestens $K = 3$ verwendet werden muss.

Bisher wurde angenommen, dass der Verallgemeinerungsfehler für den Angreifer genauso groß ist wie für die Kommunikationspartner. Dies stimmt jedoch

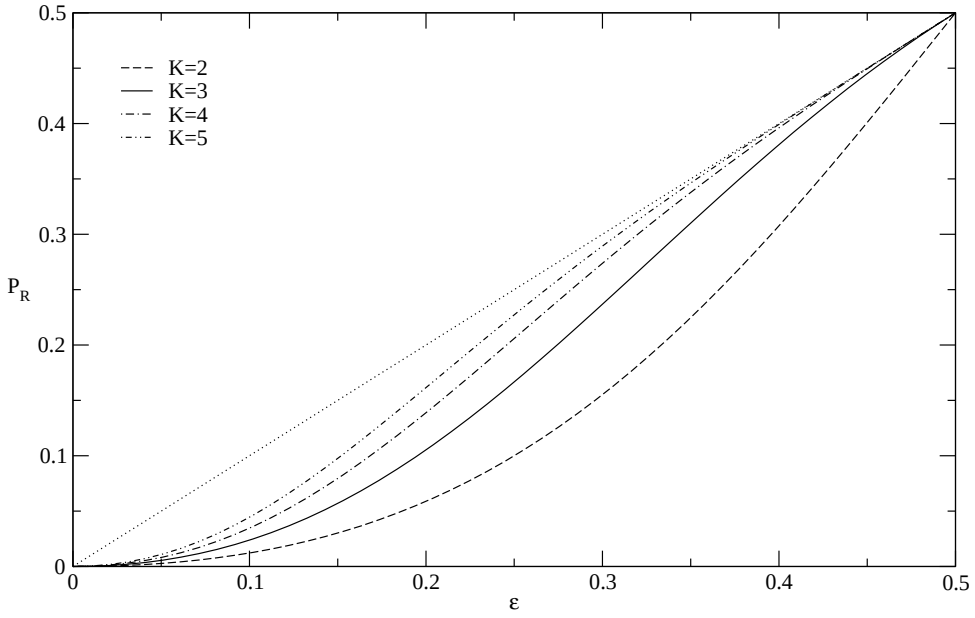


Abbildung 3.5: Wahrscheinlichkeit P_R^B für die repulsive Wirkung eines Lernschritts. Zusätzlich ist noch die entsprechende Wahrscheinlichkeit für E eingezeichnet (gepunktete Linie).

nur zu Beginn der Synchronisation. Wegen des hier beschriebenen Nachteils für E steigt nämlich der Überlapp ρ^{AE} langsamer als ρ^{AB} an. Dadurch wird im weiteren Verlauf $\epsilon^{AE} > \epsilon^{AB}$ gelten, so dass die Benachteiligung des Angreifers noch größer wird.

3.2.3 Vermeidung repulsiver Schritte beim Angriff

E kann aber versuchen, den Nachteil durch verbesserte Angriffsmethoden auszugleichen. Viele der in Kapitel 2.3 vorgestellten Methoden beruhen darauf, verschiedene Möglichkeiten parallel zu berücksichtigen. Hierfür sind große Rechenkapazitäten erforderlich.

Der geometrische Angriff verwendet aber einen anderen Ansatz. So versucht E repulsive Schritte zu vermeiden, indem bei Bedarf ($\tau^E \neq \tau^A = \tau^B$) der Zustand σ_i^E eines Zwischenschichtneurons geändert wird. Dies gelingt jedoch nicht immer, weil die σ_i^A bzw. σ_i^B dem Angreifer nicht bekannt sind.

Für $K = 2$ soll nun die Häufigkeit repulsiver Schritte bei einem solchen Angriff abgeschätzt werden. Dazu wird angenommen, dass ein einzelner Fehler von E mit der Wahrscheinlichkeit β behoben werden kann. Mit der Wahrscheinlichkeit $1-\beta$ bewirkt der Korrekturversuch aber, dass danach in beiden versteckten Einheiten ein repulsiver Schritt auftritt. Bei zwei Fehlern erfolgt keine Korrektur, da ja $\tau^A = \tau^B = \tau^E$ gilt. Daraus folgt für die Wahrscheinlichkeit, dass eine Änderung

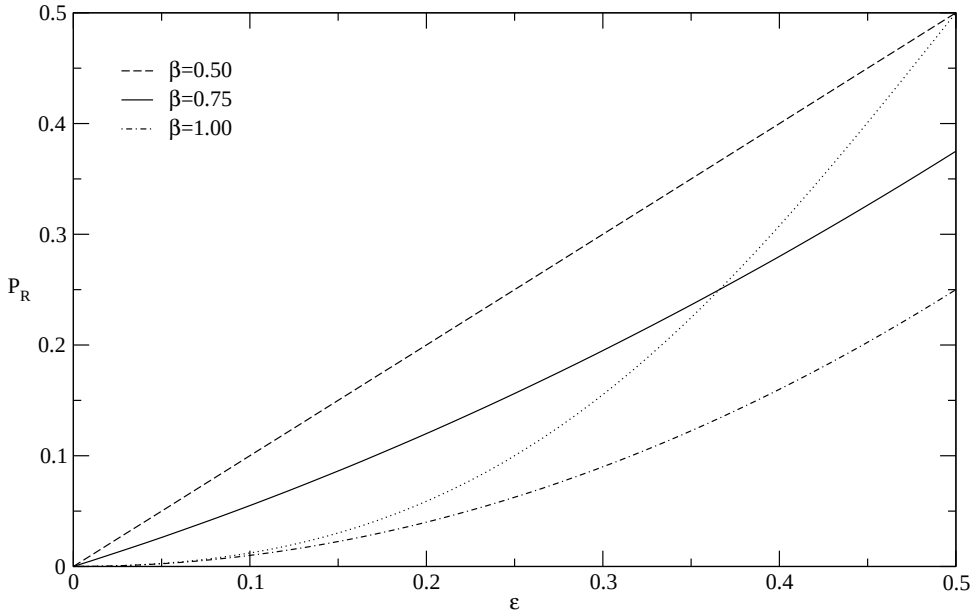


Abbildung 3.6: Wahrscheinlichkeit P_R^E für $K = 2$ bei unterschiedlicher Treffsicherheit β des Korrektur-Algorithmus. Zusätzlich ist noch P_R^B eingezeichnet (gepunktete Linie).

der Gewichte im i-ten Teilnetz der Tree Parity Machine von E repulsiv wirkt:

$$P_R^E = P(\sigma_i^E \neq \sigma_i^A | \tau^A = \tau^B) = 2(1-\beta)(1-\epsilon)\epsilon + \epsilon^2 \,. \tag{3.32}$$

Durch die effizientere Angriffsmethode sinkt also die Häufigkeit repulsiver Schritte für E ab. Dabei hängt der Erfolg von der Treffsicherheit β des Korrekturalgorithmus ab. Es kann sogar sein, dass die Wahrscheinlichkeit unter den Wert von P_R^B fällt, wie in Abbildung 3.6 zu sehen ist. Somit kann E durch die Verwendung einer guten Angriffsmethode seinen Nachteil gegenüber A und B kompensieren. Dies erklärt auch den Erfolg des geometrischen Angriffs.

3.3 Synchronisationszeit

Die vollständige Synchronisation der Tree Parity Machines von A und B ist erreicht, wenn die Bedingung $w_{ij}^A = w_{ij}^B$ für alle $i \in \{1, \ldots, K\}$ und $j \in \{1, \ldots, N\}$ erfüllt ist. Als Synchronisationszeit t_{sync} definiert man die Zahl der Lernschritte bis zum Erreichen dieses Zustands. Allerdings läuft der Schlüsselaustausch nicht immer gleich ab. Dies ist bereits in Abbildung 3.2 an der auftretenden Standardabweichung zu erkennen. Denn sowohl die Eingaben x_{ij} als auch die anfänglichen Gewichte $w_{ij}^{A/B}$ werden zufällig gewählt. Deshalb ist die Synchronisationszeit ebenfalls eine Zufallsvariable. Im Folgenden soll ihr Erwartungswert $\langle t_{sync} \rangle$ untersucht werden.

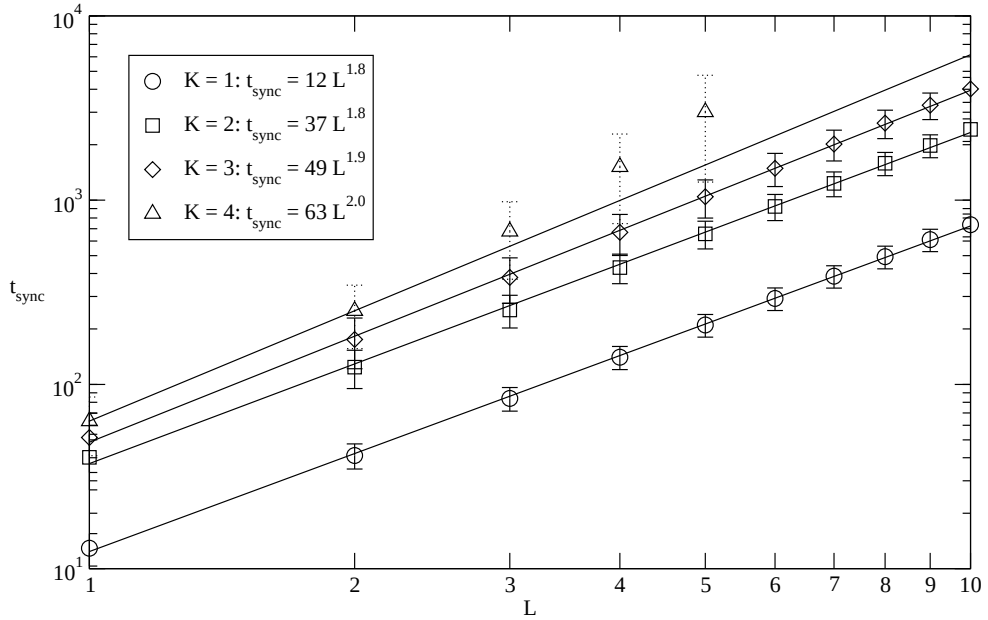


Abbildung 3.7: Synchronisationszeit bei Verwendung der Hebbschen Lernregel für $N = 1000$, gemittelt über 10 000 Simulationen. Die Fehlerbalken geben die Standardabweichung von t_{sync} an.

Für $K = 1$ können keine repulsiven Schritte auftreten. Vernachlässigt man auch noch den Einfluss der Lernregel, so ist die Bewegung zweier korrespondierender Gewichte w_{ij}^A und w_{ij}^B unabhängig von den anderen und kann als Random Walk mit reflektierenden Randbedingungen angesehen werden. Wie in [12] angegeben, folgt daraus für die Synchronisationszeit:

$$\langle t_{sync} \rangle \propto L^2 \ln N \,. \tag{3.33}$$

Betrachtet man nun zwei Tree Parity Machines mit $K > 1$, so können auch repulsive Schritte auftreten. Abbildung 3.7 zeigt aber, dass das grundlegende Skalenverhalten gleich bleibt. Diese Beschreibung ist jedoch nur richtig, solange die Bedingung $L < O(\sqrt{N})$ erfüllt ist [9]. Bei einer Erhöhung von K wird der Gültigkeitsbereich für Gleichung (3.33) kleiner. Gleichzeitig wächst der Proportionalitätsfaktor an.

Wenn $L < O(\sqrt{N})$ gilt, sind auch die Unterschiede zwischen den Lernregeln gering. Für große L zeigen sich aber Abweichungen, wie in Abbildung 3.8 zu sehen ist. Diese werden durch das lokale Feld hervorgerufen, dessen Beitrag in den Lernregeln für $L \ll \sqrt{N}$ vernachlässigt werden kann (siehe Kapitel 3.1.5).

Bei Verwendung der Anti-Hebb-Lernregel bewirkt das lokale Feld eine Verkürzung des Gewichtsvektors. Deshalb stoßen die Gewichte seltener an die Ränder des Wertebereichs, so dass die Synchronisation verlangsamt wird. Dadurch kommt es zu einem stärkeren Anstieg von $\langle t_{sync} \rangle$. Die Abweichung von Gleichung (3.33) ist für $L > 5$ in Abbildung 3.8 gut zu erkennen.

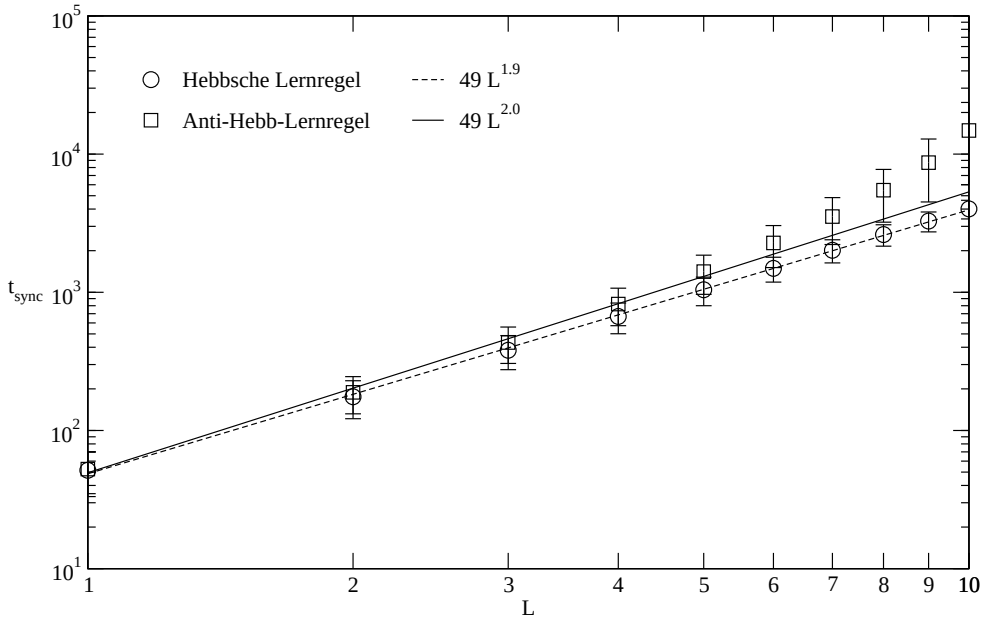


Abbildung 3.8: Abhängigkeit der Synchronisationszeit von der verwendeten Lernregel. Es wurde über 10 000 Simulationen mit den Parametern $K = 3$ und $N = 1000$ gemittelt.

Die Hebbsche Lernregel dagegen unterstützt die Synchronisation durch eine Verlängerung des Gewichtsvektors. Wenn sie verwendet wird, gilt $\langle t_{sync} \rangle \propto L^2$ auch noch für größere L.

3.4 Verteilung der Gewichte

Zu Beginn der Synchronisation sind die Gewichte in allen beteiligten neuronalen Netzen zufällig gewählt. Deshalb kommt jede Konfiguration der w_{ij} mit gleicher Wahrscheinlichkeit vor. Ein Angreifer kann zu diesem Zeitpunkt nur alle $(2L+1)^{KN}$ Möglichkeiten durchprobieren, was aussichtslos erscheint.

Im Verlauf der Synchronisation werden die Gewichte aber durch die Anwendung der Lernregeln verändert. Dabei ist es denkbar, dass die w_{ij} bestimmte Werte bevorzugt annehmen. Das wäre natürlich ein Vorteil für E, weil bei der Suche nach dem Schlüssel nur noch der Teil der Konfigurationen durchprobiert werden muss, der mit hoher Wahrscheinlichkeit vorkommt. Auch statistische Angriffsmethoden könnten dann zu einem Erfolg führen [4]. Um diese potentielle Schwäche des Schlüsselaustauschprotokolls ausschließen zu können, wird im Folgenden die Verteilung der Gewichte im Verlauf der Synchronisation untersucht.

Zu diesem Zweck bestimmt man in jedem Schritt die relative Häufigkeit p_l der Gewichte w_{ij}^A mit Wert $l \in \{-L, \ldots, +L\}$. Daraus kann dann die Entropie S

der Gewichtsverteilung berechnet werden [15]:

$$S = -\sum_{l=-L}^{+L} p_l \ln p_l \,. \tag{3.34}$$

Falls die Verteilung der Gewichte völlig zufällig ist und den gesamten Wertebereich umfasst, gilt $p_l = (2L+1)^{-1}$ für alle l. In diesem Fall erreicht die Entropie ihren Maximalwert $S_{max} = \ln(2L+1)$. Je kleiner jedoch S ist, desto eher kann ein Angreifer die ungleiche Verteilung der Gewichte ausnutzen.

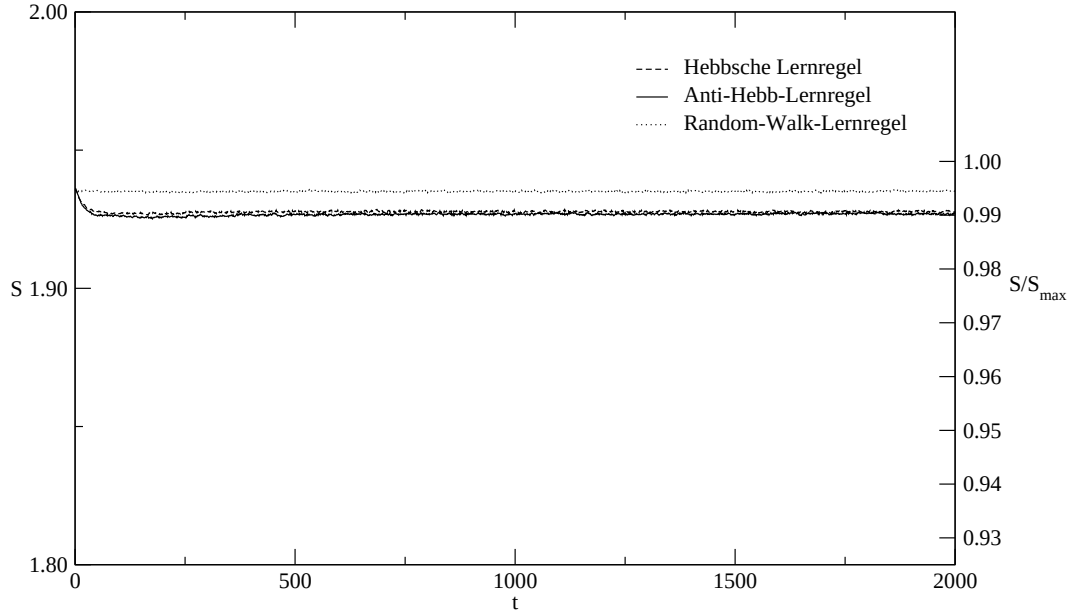


Abbildung 3.9: Entropie der Gewichtsverteilung bei der Synchronisation zweier Tree Parity Machines, gemittelt über 1000 Simulationen mit $K = 3$, $L = 3$ und $N = 100$.

In Abbildung 3.9 ist jedoch zu sehen, das diese Befürchtung nicht zutrifft. Unabhängig von der Wahl der Lernregel fällt der Mittelwert $\langle S \rangle$ der Entropie nicht unter $0.99\, S_{max}$ ab. Somit sind die Gewichte auch nach vielen Synchronisationsschritten noch annähernd gleichverteilt. Da die Ausgabe τ bei der Random-Walk-Lernregel keinen direkten Einfluss auf die Richtung der Gewichtsanpassungen hat, liegt die Entropie in diesem Fall höher. Bei den beiden anderen Lernregeln ändert sich dagegen die Länge des Gewichtsvektors. Dadurch werden bestimmte Werte der w_{ij} minimal bevorzugt, was das Absinken der Entropie zu Beginn der Synchronisation erklärt.

E kann also keinen Vorteil aus der Verteilung der Gewichte erlangen. Somit ist der neuronale Schlüsselaustausch gegenüber statistischen Angriffen geschützt. Der Angreifer muss deshalb andere Methoden einsetzen, wenn er erfolgreich sein will.

3.5 Lernen eines Angreifers

Die aussichtsreichsten Angriffsmethoden beruhen darauf, dass E ebenfalls neuronale Netze verwendet. Dabei dienen die zwischen A und B ausgetauschten Informationen, die Eingaben x_{ij} sowie die Ausgaben τ^A und τ^B, als Beispiele für ein Online-Lernverfahren. Allerdings ist das Lernen erheblich langsamer als die Synchronisation, weil hier mehr repulsive Schritte auftreten. Deshalb ist ein Erfolg nur möglich, wenn der Angreifer zusätzliche Algorithmen einsetzt, um diesen Nachteil auszugleichen.

Die Sicherheit des neuronalen Schlüsselaustauschs hängt also davon ab, ob es den Kommunikationspartnern gelingt, die vollständige Synchronisation schneller als der Angreifer zu erreichen. In diesem Fall ist es möglich, den Austausch der Ausgaben $\tau^{A/B}$ zu beenden, bevor E die Gewichte ermitteln kann.

Ein Angriff wird als erfolgreich gewertet, wenn es E gelingt, zum Zeitpunkt der vollständigen Synchronisation von A und B eine Übereinstimmung von mindestens 98% der Gewichte zu erreichen [9]. Durch diese Definition werden die Fluktuationen in den Simulationen reduziert. Denn ähnlich wie die Synchronisationszeit ist auch der Erfolg eines Angreifers von den Anfangsbedingungen und den zufällig gewählten Eingaben abhängig. Deshalb kann für jede Angriffsmethode auch nur eine Erfolgswahrscheinlichkeit P_E angegeben werden.

3.5.1 Verlauf der Überlapps

Die Ausgangssituation beim neuronalen Schlüsselaustausch ist für Kommunikationspartner und Angreifer gleich. Denn die Gewichte der Tree Parity Machines werden zufällig und unkorreliert gewählt, so dass das Skalarprodukt zweier verschiedener Gewichtsvektoren den Erwartungswert $\langle \vec{w}_i^m \cdot \vec{w}_i^n \rangle = 0$ hat. Allerdings kann E im Gegensatz zu A und B keinen Einfluss auf den Verlauf der Synchronisation nehmen. Dadurch kommt es zu der in Kapitel 3.2 beschriebenen Benachteiligung des Lernens gegenüber der Synchronisation, die sich auch im Verlauf des Überlapps äußert.

Für $K > 1$ besteht jede Tree Parity Machine aus mehreren versteckten Einheiten. Entsprechend können auch K verschiedene ρ_i^{mn} angegeben werden, die den Synchronisationsgrad jeweils korrespondierender versteckter Einheiten in zwei neuronalen Netzen beschreiben. Da die einzelnen Zwischenschichtneuronen keine Besonderheiten aufweisen, haben die ρ_i^{mn} ähnliche Werte. Deshalb ist es sinnvoll, zur Beschreibung des Synchronisationsgrads den Mittelwert ρ^{mn} zu verwenden:

$$\rho^{mn} = \frac{1}{K} \sum_{i=1}^{K} \rho_i^{mn} \,. \tag{3.35}$$

Der typische Verlauf von ρ^{AB} und ρ^{AE} ist in Abbildung 3.10 dargestellt. Dabei wurde neben dem Lernen als einfachster Angriffsmethode auch der geometrische Angriff berücksichtigt.

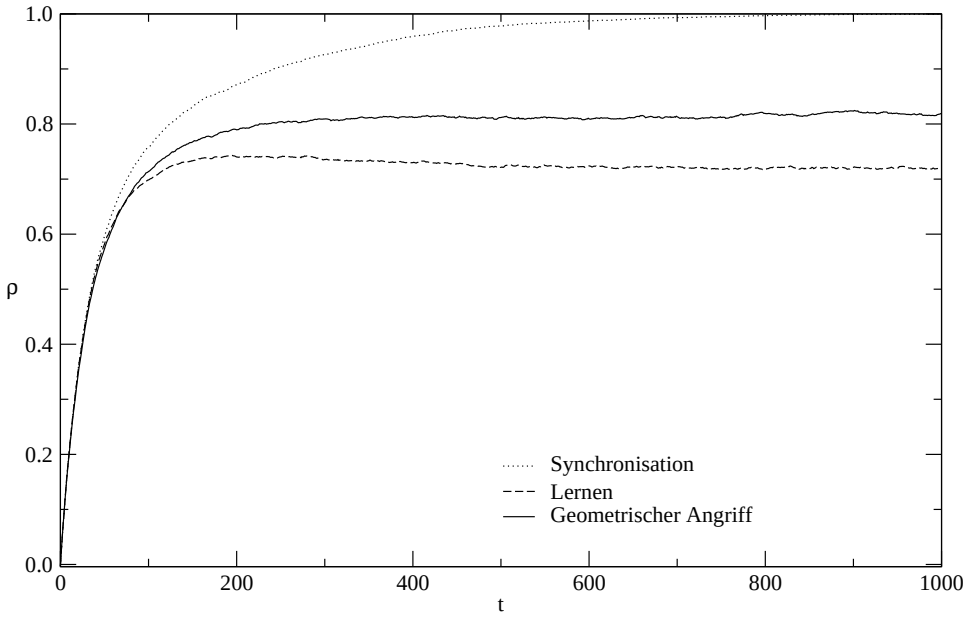


Abbildung 3.10: Verlauf des Überlapps für die Parameter $K = 3$, $L = 3$ und $N = 100$. Es wurde über 1000 Simulationen gemittelt und die Anti-Hebb-Lernregel verwendet.

Bis zu einem Überlapp von $\rho = 0.5$ ist kein Unterschied zwischen dem Lernen von E und der Synchronisation von A und B feststellbar. Danach aber steigt ρ^{AE} langsamer an als ρ^{AB}. Es ist aber zu beachten, dass im Diagramm der Mittelwert des Überlapps über mehrere Simulationen aufgetragen ist. Es gibt also sowohl Angreifer, die einen besseren Überlapp bis hin zur vollständigen Synchronisation erreichen, als auch solche, bei denen ρ^{AE} kleiner ist.

Dabei zeigt sich, dass der geometrische Angriff durch seinen Korrekturalgorithmus eine wesentliche Verbesserung von $\langle \rho^{AE} \rangle$ erreicht. Dies wirkt sich in entsprechender Weise auch auf die Erfolgswahrscheinlichkeit aus. Für die in Abbildung 3.10 verwendeten Parameter hat der geometrische Angriff beispielsweise eine Erfolgswahrscheinlichkeit von $P_E = 0.237$. Dagegen konnte beim einfachen Lernen der Ausgaben in 1000 Simulationen kein Erfolg festgestellt werden. In beiden Fällen wurde von E nur eine Tree Parity Machine eingesetzt.

3.5.2 Geometrischer Angriff

Die Sicherheit des neuronalen Schlüsselaustauschs ist nur dann gewährleistet, wenn die Erfolgswahrscheinlichkeit P_E für jede Angriffsmethode vernachlässigbar klein ist. Wie auch bei anderen kryptographischen Verfahren hängt der gerade noch tolerable Maximalwert für P_E davon ab, welche Anforderungen an den Schutz der zu übermittelnden Nachricht gestellt werden [4]. Wenn A und B ein Sicherheitsniveau erreichen wollen, das einer Verschlüsselung mit einem 128 bit-

Schlüssel entspricht, so müssen sie durch geeignete Wahl der Parameter für alle Angriffsmethoden $P_E < 2^{-128}$ erreichen.

Dabei ist vor allem auf den geometrischen Angriff zu achten, da E mit dieser Methode bereits bei minimalem Aufwand eine hohe Erfolgswahrscheinlichkeit erreichen kann. Eine Vergrößerung von L erschwert diesen Angriff, weil dann mehr Gewichtskonfigurationen möglich sind. In [9] wurde festgestellt, dass P_E exponentiell mit L abfällt, wenn für die Synchronisation die Hebbsche Lernregel verwendet wird:

$$P_E \propto e^{-yL} \,. \tag{3.36}$$

Die Konstante y hängt natürlich noch von den beiden anderen Parametern K und N ab. Wie in Abbildung 3.11 zu sehen ist, hat die Zahl der versteckten Einheiten einen sehr großen Einfluss auf die Sicherheit des neuronalen Schlüsselaustauschs.

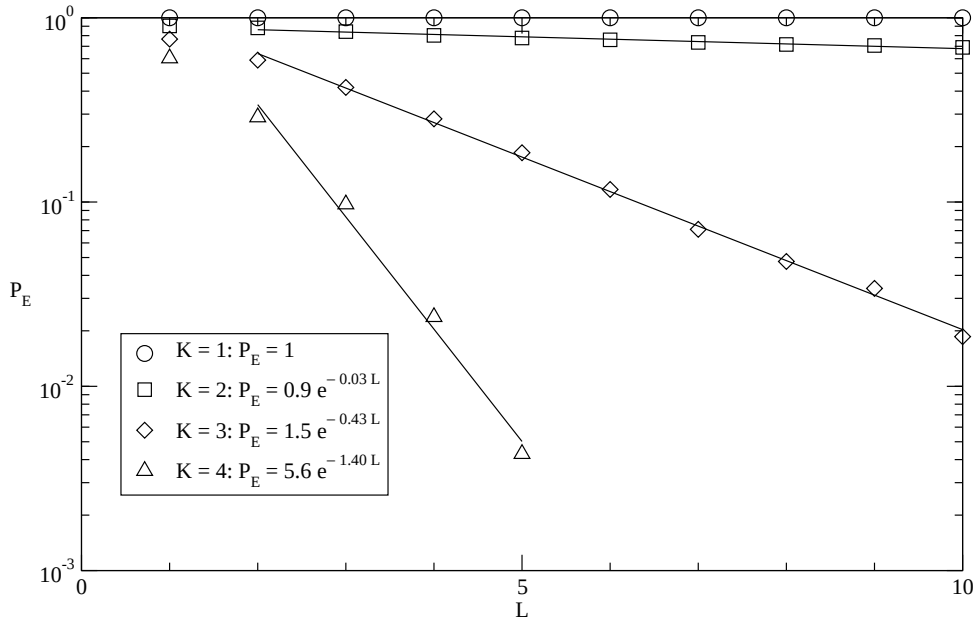


Abbildung 3.11: Erfolgswahrscheinlichkeit des geometrischen Angriffs bei Verwendung der Hebbschen Lernregel, berechnet aus 10 000 Simulationen mit $N = 1000$.

Für $K = 1$ ist ein Angriff in jedem Fall erfolgreich, weil keine repulsiven Schritte auftreten. Auch mit zwei Zwischenschichtneuronen ist der neuronale Schlüsselaustausch unsicher, da eine Erhöhung von L kaum Einfluss auf P_E zeigt. Erst mit mindestens 3 Neuronen in der Zwischenschicht lässt sich die Sicherheit merklich durch eine Erhöhung von L verbessern. Extrapoliert man den in Abbildung 3.11 dargestellten Verlauf, so wird $P_E < 2^{-128}$ für $K = 3$ und $L > 207$ erreicht. Bei vier versteckten Einheiten ist für dasselbe Sicherheitsniveau nur $L > 64$ nötig.

Auch t_{sync} hängt von L ab. Deshalb bewirkt ein größeres L nicht nur eine Verbesserung der Sicherheit, sondern auch eine Erhöhung der Synchronisationszeit. Allerdings wächst der Aufwand der Kommunikationspartner nur proportional zu

L^2, während die Komplexität eines erfolgversprechenden Angriffs exponentiell mit L zunimmt. Denn dem Angreifer bleibt nur die Möglichkeit, die abnehmende Erfolgswahrscheinlichkeit durch eine größere Zahl an neuronalen Netzen zu kompensieren. Deshalb ist der neuronale Schlüsselaustausch bei großem L sicher gegen einen geometrischen Angriff [9].

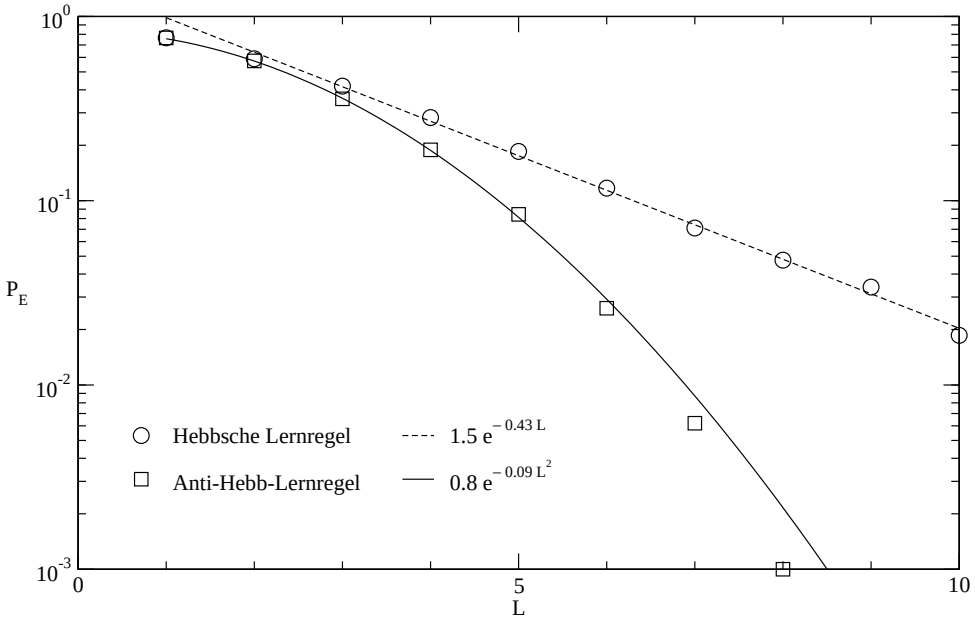


Abbildung 3.12: Abhängigkeit der Erfolgswahrscheinlichkeit des geometrischen Angriffs von der verwendeten Lernregel. P_E wurde aus 10 000 Simulationen mit den Parametern $K = 3$ und $N = 1000$ bestimmt.

Abbildung 3.12 zeigt, dass bei Anwendung der Anti-Hebb-Lernregel und $N = 1000$ Gleichung (3.36) nicht gilt. Stattdessen findet man folgenden funktionalen Zusammenhang:

$$P_E \propto e^{-uL^2} . \tag{3.37}$$

Die Ursache für das abweichende Skalenverhalten liegt in der geometrischen Arbeitsweise der Angriffsmethode. Wie in Kapitel 2.3.5 erläutert, definieren die Eingabevektoren $\vec{x}_i$ Hyperebenen X_i im Konfigurationsraum der Gewichte. Der Angreifer nimmt an, dass eine versteckte Einheit mit kleinem Abstand $d(W_i^E, X_i)$ zwischen Gewichtskonfiguration W_i^E und zugehöriger Hyperebene X_i mit hoher Wahrscheinlichkeit eine falsche Ausgabe τ^E verursacht hat. Die Anti-Hebb-Lernregel bewirkt aber eine Verkürzung der Gewichtsvektoren $\vec{w}_i$, so dass die W_i näher am Ursprung liegen. Dadurch sind die Abstände $d(W_i^E, X_i)$ – im Gegensatz zur Hebbschen Lernregel – auch dann klein, wenn W_i^A, W_i^B und W_i^E auf einer Seite der Hyperebene X_i liegen. Dies stört den Korrekturmechanismus des geometrischen Angriffs, so dass E mehr repulsive Schritte erleidet.

Für sehr große N sollten diese Unterschiede aber geringer werden. Denn für $L \ll \sqrt{N}$ hat die Form der Lernregel keinen Einfluss mehr auf die Länge des Gewichtsvektors (siehe Kapitel 3.1.5). In diesem Grenzfall ist somit eine einheitliche Erfolgswahrscheinlichkeit des geometrischen Angriffs für alle Lernregeln zu erwarten.

3.5.3 Genetischer Angriff

Alternativ zum geometrischen Angriff kann E auch den in Kapitel 2.3.4 beschriebenen genetischen Angriff verwenden. Da bei dieser Methode erheblich mehr neuronale Netze eingesetzt werden, ist auch eine größere Erfolgswahrscheinlichkeit P_E zu erwarten.

Abbildung 3.13 zeigt jedoch, dass dies nur zutrifft, wenn L klein ist. Bereits für $K = 3$, $L = 3$ und $N = 1000$ ist die Erfolgswahrscheinlichkeit des genetischen Angriffs mit maximal 10 000 neuronalen Netzen geringer als P_E beim geometrischen Angriff mit *einer* Tree Parity Machine. Somit lohnt sich der höhere Aufwand der hier betrachteten Angriffsmethode nicht.

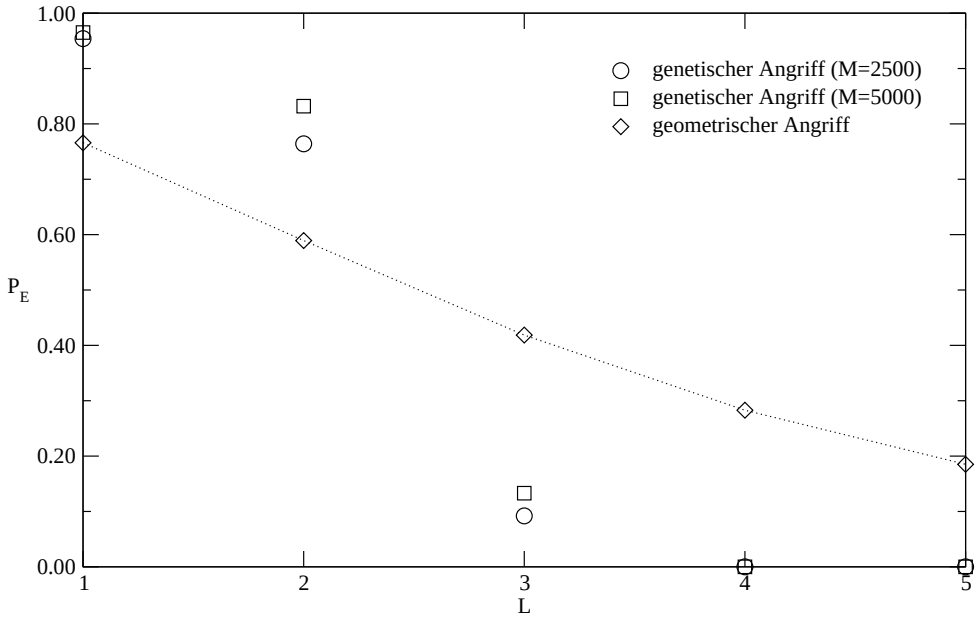


Abbildung 3.13: Erfolgswahrscheinlichkeit des genetischen Angriffs bei Verwendung der Hebbschen Lernregel, berechnet aus 1000 Simulationen mit $K = 3$ und $N = 1000$. Zum Vergleich ist die entsprechende Wahrscheinlichkeit für den geometrischen Angriff eingezeichnet.

3.6 Iterative Rechnung für $N \to \infty$

Bisher wurde der neuronale Schlüsselaustausch in relativ kleinen Systemen ($N = 100$ bzw. $N = 1000$) untersucht. Hier konnten die Synchronisationszeit t_{sync} und die Erfolgswahrscheinlichkeit P_E eines Angreifers gut durch Simulationen ermittelt werden. Bei kleinem N ist es aber schwierig, die Bedingung $L < O(\sqrt{N})$ einzuhalten. Deshalb erscheint es sinnvoll, auch Tree Parity Machines mit einer größeren Anzahl an Gewichten zu betrachten.

Simulationen sind aber nur für kleine Systeme geeignet. Denn die benötigte Rechenzeit skaliert wie $O(L^2 N \ln N)$ [9], solange $1 \ll L \ll \sqrt{N}$ gilt. Wenn diese Bedingung nicht eingehalten wird, ist eine noch höhere Rechenzeitkomplexität zu erwarten. Bereits für $K = 3$, $L = 5$ und $N = 10\,000$ dauert die Ermittlung von P_E mehrere Wochen. Für größere Systeme wird somit eine andere Untersuchungsmethode benötigt.

Allerdings stellt man fest, dass die Varianz des Überlapps bei der Synchronisation diskreter neuronaler Netze für $N \to \infty$ nicht verschwindet [10]. Folglich haben die Ordnungsparameter in diesem Fall keine selbstmittelnden Eigenschaften [16]. Deshalb ist auch eine direkte analytische Lösung ohne Verwendung von Zufallsvariablen nicht möglich. Stattdessen kann ein iteratives Verfahren verwendet werden, das in [11] beschrieben wird. Es erlaubt die Berechnung der Vorgänge beim neuronalen Schlüsselaustausch im Grenzfall $N \to \infty$. Mit dieser Methode soll nun das Skalenverhalten der Größen t_{sync} und P_E für große N untersucht werden.

3.6.1 Beschreibung der Synchronisation

Für die Ermittlung der Synchronisationszeit genügt es, die iterative Rechnung zunächst nur für zwei Tree Parity Machines durchzuführen. Denn ein Angreifer hat keinen Einfluss auf den Verlauf der Synchronisation von A und B. Um später auch Angriffe untersuchen zu können, ist dann eine Erweiterung auf drei neuronale Netze nötig.

Die Gewichtskonfiguration zweier Tree Parity Machines wird bei der iterativen Rechnung durch je eine $(2L+1) \times (2L+1)$-Matrix $\mathcal{F}_i$ pro versteckter Einheit beschrieben. Das Matrixelement $f_{ab}^i := (\mathcal{F}_i)_{ab}$ gibt dabei die relative Häufigkeit an, mit der Gewichte mit $w_{ij}^A = a$ und $w_{ij}^B = b$ im i-ten Teilnetz der Tree Parity Machines vorkommen. Für die Indizes gilt dabei $a, b \in \{-L, \ldots, +L\}$. Diese Darstellung ist unabhängig von N und kann somit auch im Grenzfall $N \to \infty$ verwendet werden. Zu Beginn der Synchronisation werden alle Matrixelemente f_{ab}^i auf den Wert $(2L+1)^{-2}$ gesetzt. Das entspricht einer unkorrelierten Gleichverteilung aller Gewichte $w_{ij}^{A/B}$.

Für die Beschreibung der Synchronisation werden natürlich die Ordnungsparameter Q_i^A, Q_i^B und R_i^{AB} benötigt, die sich aber leicht aus der Matrix $\mathcal{F}_i$

berechnen lassen:

$$Q_i^A = \sum_{a=-L}^{L} \sum_{b=-L}^{L} a^2 f_{ab}^i \,; \tag{3.38}$$

$$Q_i^B = \sum_{a=-L}^{L} \sum_{b=-L}^{L} b^2 f_{ab}^i \,; \tag{3.39}$$

$$R_i^{AB} = \sum_{a=-L}^{L} \sum_{b=-L}^{L} ab f_{ab}^i \,. \tag{3.40}$$

Aus diesen Größen kann dann der Überlapp ρ_i^{AB} berechnet werden. Dazu wird Gleichung (3.3) verwendet.

In jedem Synchronisationsschritt werden zunächst die Ausgaben der versteckten Einheiten bestimmt. Betrachtet man zunächst nur eine Tree Parity Machine, so treten die beiden möglichen Zustände $\sigma_i = +1$ und $\sigma_i = -1$ eines Zwischenschichtneurons mit gleicher Häufigkeit auf. Bei der iterativen Rechnung wird deshalb ein Pseudozufallszahlengenerator verwendet, um eine der 2^K verschiedenen Kombinationen der σ_i^A mit gleicher Wahrscheinlichkeit auszuwählen. Die σ_i^B hängen dann vom jeweiligen Überlapp ρ_i^{AB} ab. Denn mit Gleichung (3.28) ergibt sich hieraus der Verallgemeinerungsfehler ϵ_i^{AB}, der die Wahrscheinlichkeit angibt, dass $\sigma_i^B \neq \sigma_i^A$ ist. Deshalb wird für jede versteckte Einheit eine Zahl p_i zufällig und gleichverteilt aus dem Einheitsintervall gezogen. Für $p_i \leq \epsilon_i^{AB}$ wird $\sigma_i^B = -\sigma_i^A$ gesetzt; andernfalls gilt $\sigma_i^B = \sigma_i^A$. Die Gesamtausgaben τ der Tree Parity Machines errechnen sich dann wie bisher als Produkt der σ_i. Nur wenn $\tau^A = \tau^B$ ist, werden im jeweiligen Synchronisationsschritt die Matrixelemente f_{ab}^i verändert.

Wenn $\tau^A = \tau^B = \sigma_i^A = \sigma_i^B$ gilt, findet in der i-ten versteckten Einheit ein attraktiver Lernschritt statt. Für $N \to \infty$ ist dabei über die Eingaben x_{ij} in der verwendeten Lernregel zu mitteln. Die sich daraus ergebenden neuen Matrixelemente f_{ab}^{i+} sind für alle drei möglichen Lernregeln gleich:

$$f_{a,b}^{i+} = \frac{1}{2}(f_{a+1,b+1}^i + f_{a-1,b-1}^i) \,. \tag{3.41}$$

Diese Formel gilt aber nur für $-L < a, b < +L$. Für die Randelemente der Matrix müssen zusätzlich die reflektierenden Randbedingungen beim neuronalen Schlüsselaustausch berücksichtigt werden. Dabei ergeben sich ähnliche Gleichungen, die in Anhang B angegeben sind.

Für $\tau^A = \tau^B$ erfolgt ein repulsiver Schritt dann, wenn $\sigma_i^A \neq \sigma_i^B$ ist. Dabei ändern sich die Gewichte der i-ten versteckten Einheit nur im neuronalen Netz eines Partners. Falls $\tau^B \neq \sigma_i^B$ gilt, werden nur die Gewichte von A angepasst:

$$f_{a,b}^{i+} = \frac{1}{2}(f_{a+1,b}^i + f_{a-1,b}^i) \,. \tag{3.42}$$

Für $\tau^A \neq \sigma_i^A$ dagegen ändern sich nur die Gewichte von B:

$$f_{a,b}^{i+} = \frac{1}{2}(f_{a,b+1}^i + f_{a,b-1}^i)\,. \tag{3.43}$$

In beiden Fällen gelten die angegebenen Gleichungen aber nicht für Matrixelemente, die am Rand liegen (siehe Anhang B).

Im Grenzfall $N \to \infty$ wird eine vollständige Synchronisation erst nach unendlich vielen Schritten erreicht. Um dennoch das Skalenverhalten von t_{sync} und P_E untersuchen zu können, muss ein Synchronisationskriterium festgelegt werden. Hierfür wird wie in [11] die Bedingung $\rho \geq 0.9$ verwendet.

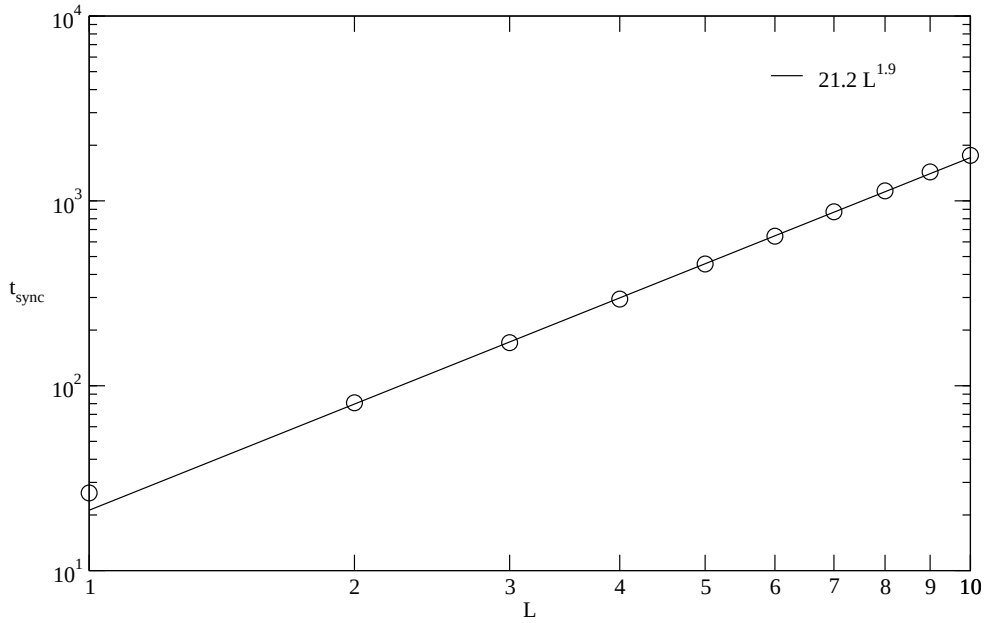


Abbildung 3.14: Zahl der benötigten Schritte bis zum Erreichen des Synchronisationskriteriums $\rho \geq 0.9$, gemittelt über 10 000 Berechnungen für $K = 3$.

Abbildung 3.14 zeigt, dass die mittlere Zahl der Schritte bis zur Synchronisation von A und B proportional zu L^2 ansteigt. Dieses Verhalten wurde auch in den Simulationen beobachtet. Die in der iterativen Rechnung ermittelten Werte für t_{sync} hängen aber vom gewählten Synchronisationskriterium ab. Deshalb liefert die iterative Rechnung einen kleineren Proportionalitätsfaktor als die Simulationen zu $N = 1000$.

3.6.2 Untersuchung des geometrischen Angriffs

Für die Untersuchung des geometrischen Angriffs muss die iterative Rechnung auf drei beteiligte neuronale Netze erweitert werden. Dazu werden die Matrizen durch Tensoren 3. Stufe ersetzt. Das Element $f_{abe}^i := (\mathcal{F}_i)_{abe}$ gibt nun die relative

Häufigkeit an, mit der Gewichte mit $w_{ij}^A = a$, $w_{ij}^B = b$ und $w_{ij}^E = e$ in der i-ten versteckten Einheit auftreten ($a, b, e \in \{-L, \ldots, +L\}$). Zu Beginn der Synchronisation werden alle f_{abe}^i auf den Wert $(2L+1)^{-3}$ gesetzt. Wie zuvor beschreibt der Startzustand gleichverteilte und unkorrelierte Gewichte.

Im Gegensatz zu anderen Angriffsmethoden verwendet der geometrische Angriff nicht nur die Ausgaben σ_i^E sondern auch die lokalen Felder h_i^E der Zwischenschichtneuronen. Deshalb reicht es für die iterative Rechnung nicht aus, nur die Verallgemeinerungsfehler ϵ_i^{mn} zu berücksichtigen. Stattdessen wird nun die Wahrscheinlichkeitsdichte $P(h_i^A, h_i^B, h_i^E)$ der lokalen Felder benötigt [11]:

$$P(h_i^A, h_i^B, h_i^E) = \frac{1}{\sqrt{(2\pi)^3 \det \mathcal{C}_i}} \, e^{-\frac{1}{2}(h_i^A, h_i^B, h_i^E)\, \mathcal{C}_i^{-1}\, (h_i^A, h_i^B, h_i^E)^T} . \tag{3.44}$$

Die in der Gleichung auftretende Kovarianzmatrix $\mathcal{C}_i$ beschreibt die Korrelationen zwischen den i-ten versteckten Einheiten der neuronalen Netze von A, B und E:

$$\mathcal{C}_i = \begin{pmatrix} Q_i^A & R_i^{AB} & R_i^{AE} \\ R_i^{AB} & Q_i^B & R_i^{BE} \\ R_i^{AE} & R_i^{BE} & Q_i^E \end{pmatrix} . \tag{3.45}$$

Die Matrixelemente von $\mathcal{C}_i$ lassen sich leicht aus den f_{abe}^i berechnen:

$$\begin{aligned} R_i^{AB} &= \sum_{a,b,e=-L}^{L} ab f_{abe}^i ; \qquad Q_i^A = \sum_{a,b,e=-L}^{L} a^2 f_{abe}^i ; \\ R_i^{AE} &= \sum_{a,b,e=-L}^{L} ae f_{abe}^i ; \qquad Q_i^B = \sum_{a,b,e=-L}^{L} b^2 f_{abe}^i ; \\ R_i^{BE} &= \sum_{a,b,e=-L}^{L} be f_{abe}^i ; \qquad Q_i^E = \sum_{a,b,e=-L}^{L} e^2 f_{abe}^i . \end{aligned} \tag{3.46}$$

Die Wahrscheinlichkeitsdichte $P(h_i^A, h_i^B, h_i^E)$ wird nun benutzt, um die lokalen Felder der drei Tree Parity Machines in jedem Schritt per Pseudozufallszahlengenerator zu bestimmen. Die Verwendung der *Rejection Method* [17] stellt dabei sicher, dass die Zufallszahlen die richtige Verteilung aufweisen. Allerdings wird das Auftreten sehr großer lokale Felder mit $|h| > 4\sqrt{Q}$ vernachlässigt. Dies verursacht aber keine Abweichungen von der bei Simulationen beobachteten Verteilung, da die Wahrscheinlichkeit für $|h| > 4\sigma_h$ kleiner als 10^{-4} ist.

Mit der Beziehung $\sigma_i^m = \mathrm{sign}(h_i^m)$ lassen sich anschließend die σ_i^m und die τ^m der drei Tree Parity Machines berechnen. Falls $\tau^A = \tau^B \neq \tau^E$ gilt, werden die σ_i^E entsprechend der geometrischen Angriffsmethode angepasst, so dass danach die Ausgaben τ aller Tree Parity Machines übereinstimmen.

Für die Berechnung der neuen f_{abe}^{i+} müssen zunächst die Neuronen der Zwischenschicht ermittelt werden, deren Gewichte sich ändern. Zu diesem Zweck

werden die binären Hilfsgrößen Δ_i^m für alle $i \in \{1, \ldots, K\}$ und $m \in \{A, B, E\}$ berechnet:

$$\Delta_i^m = \Theta(\sigma_i^m \tau^A)\Theta(\tau^A \tau^B)\,. \tag{3.47}$$

Dann lässt sich die Berechnung der neuen Nicht-Randelemente f_{abe}^{i+}, die die Bedingung $-L < a, b, e < +L$ erfüllen, in einer Formel zusammenfassen:

$$f_{a,b,e}^{i+} = \frac{1}{2}\left(f^i_{a+\Delta_i^A, b+\Delta_i^B, e+\Delta_i^E} + f^i_{a-\Delta_i^A, b-\Delta_i^B, e-\Delta_i^E}\right)\,. \tag{3.48}$$

Für die Randelemente ergeben sich ähnliche Gleichungen, in denen aber zusätzlich die Randbedingung $|w_{ij}| \leq L$ berücksichtigt wird.

Zur Bestimmung des Skalenverhaltens von P_E gilt ein Angriff als erfolgreich, wenn $\rho^{AE} \geq 0.9$ oder $\rho^{BE} \geq 0.9$ zu einem früheren Zeitpunkt erreicht wird als das Synchronisationskriterium $\rho^{AB} \geq 0.9$.

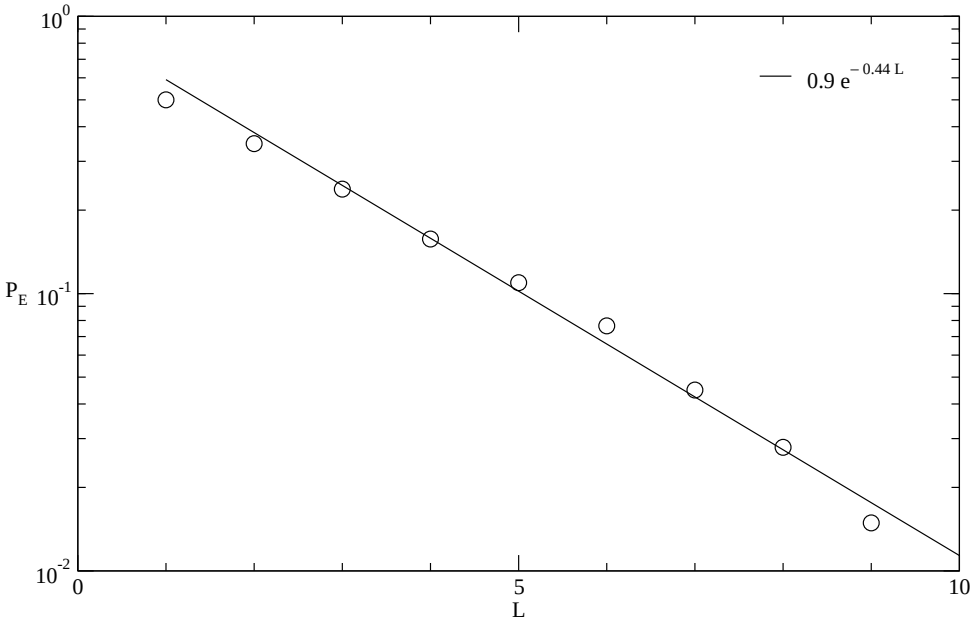


Abbildung 3.15: Erfolgswahrscheinlichkeit des geometrischen Angriffs, ermittelt aus 10 000 Berechnungen für $K = 3$.

In Abbildung 3.15 ist zu sehen, dass die Erfolgswahrscheinlichkeit eines geometrischen Angriffs exponentiell mit zunehmendem L abfällt. Dieses Verhalten stimmt mit den Simulationsergebnissen bei Verwendung der Hebbschen Lernregel überein. Somit gilt Gleichung (3.36) für $N \to \infty$ unabhängig von der Lernregel.

Kapitel 4

Zeitreihenerzeugung mit der Verwirrten Tree Parity Machine

Nach dem neuronalen Schlüsselaustausch stimmen die Gewichte in den Tree Parity Machines von A und B überein. Wie bereits in Kapitel 2 erwähnt, können die beiden Partner nun ein symmetrisches Verschlüsselungsverfahren bei der Übermittlung der Nachricht einsetzen. Der benötigte gemeinsame Schlüssel wird dazu aus den w_{ij} berechnet.

Viele Verschlüsselungsverfahren verwenden einen Pseudozufallszahlengenerator, um eine Bitsequenz zu erzeugen. Der Schlüssel dient dabei als Startwert für den Generator, so dass auf diese Weise bei A und B dieselbe Zeitreihe erzeugt wird. Der Sender verschlüsselt, indem er die Exklusiv-ODER-Verknüpfung der Bitsequenz mit der Binärdarstellung seiner Nachricht berechnet. Zum Entschlüsseln muss der Empfänger diesen Vorgang nur noch einmal durchführen [4].

Solche Zeitreihen lassen sich auch mit einer Tree Parity Machine erzeugen. Das hat beim Schlüsselaustausch den Vorteil, dass die neuronalen Netze nach der Synchronisation auch gleich für die Verschlüsselung bzw. Entschlüsselung der Nachricht verwendet werden können. Allerdings reicht es für einen Angreifer aus, den Inhalt der Nachricht teilweise zu kennen, um daraus einen Ausschnitt der benutzten Bitsequenz zu berechnen. Dieser *known-plaintext Angriff* ermöglicht eine nachträgliche Synchronisation der Tree Parity Machine von E, wenn auch beim Verschlüsseln öffentliche Eingaben x_{ij} verwendet werden. Deshalb werden die neuronalen Netze von A und B als Verwirrte Tree Parity Machines betrieben, die sich ihre Eingaben über einen Feedback-Mechanismus selbst erzeugen.

Die Sicherheit der Verschlüsselung hängt aber auch von den Eigenschaften der verwendeten Zeitreihe ab. Deshalb wurden die Pseudozufallszahlengeneratoren der üblichen Verschlüsselungsverfahren gründlich auf ihre kryptographische Eignung getestet. Hier soll nun die Zeitreihenerzeugung mit der Verwirrten Tree Parity Machine in ähnlicher Weise untersucht werden.

4.1 Erzeugung von Bitsequenzen

Bei einem Feedforward-Netz hängt die Ausgabe nur von der aktuellen Eingabe ab. Deshalb benötigt man für die Zeitreihenerzeugung einen zusätzlichen Mechanismus, der in jedem Schritt eine neue Eingabe generiert. Zu diesem Zweck bietet es sich an, die Zustandsfolge des neuronalen Netzes zu speichern und den Eingangsneuronen zur Verfügung zu stellen. Das Prinzip eines solchen Feedback-Mechanismus soll zunächst an einem Perzeptron gezeigt werden. Anschließend kann es leicht auf eine Tree Parity Machine übertragen werden.

4.1.1 Das Perzeptron als Zeitreihengenerator

Bereits mit dem einfachsten neuronalen Netz, dem diskreten Perzeptron, lassen sich Bitsequenzen erzeugen. Als Anfangsbedingungen sind ein N-dimensionaler Gewichtsvektor $\vec{w}(0)$ und eine Anfangsbitfolge vorzugeben.

- Der *Bit-Generator* [18] verwendet die letzten N Elemente der Zeitreihe als Eingabe. Das nächste Element $S(t)$ entspricht dann der Ausgabe des Perzeptrons:

$$S(t) = \text{sign}\left(\sum_{j=1}^{N} w_j S(t-j)\right) . \tag{4.1}$$

  Der Gewichtsvektor des Bit-Generators ist fest vorgegeben, so dass immer $\vec{w}(t) = \vec{w}(0)$ gilt.

- Auch der *Verwirrte Bit-Generator* [5] erzeugt eine Zeitreihe und nutzt diese gleichzeitig als Eingabe. Allerdings wird nun als nächstes Element $S(t)$ das Gegenteil der aktuellen Ausgabe verwendet:

$$S(t) = -\text{sign}\left(\sum_{j=1}^{N} w_j(t) S(t-j)\right) . \tag{4.2}$$

  Zusätzlich versucht das neuronale Netz, die erzeugte Bitsequenz zu lernen, so dass sich der Gewichtsvektor $\vec{w}(t)$ in jedem Schritt ändert:

$$w_j(t+1) = w_j(t) + S(t)S(t-j) . \tag{4.3}$$

  Im Unterschied zu den bisher betrachteten neuronalen Netzen ist der Wertebereich der Gewichte beim Verwirrten Bit-Generator nicht beschränkt.

Bei der Untersuchung der Zeitreihen, die von den beiden neuronalen Netzen erzeugt werden, stellt man fest, dass deutliche Abweichungen von den Eigenschaften einer zufälligen Bitsequenz auftreten. So kann ein weiteres Perzeptron die Ausgabenfolge des Bit-Generators leicht lernen und anschließend richtig vorhersagen [18]. Deshalb ist dieses neuronale Netz für kryptographische Zwecke ungeeignet.

Dagegen kann die Zeitreihe des Verwirrten Bit-Generators von einem Perzeptron mit der gleichen Anzahl Gewichte deutlich schlechter vorhergesagt werden. Insbesondere ist die Ausgabe des Schüler-Netzes immer falsch, wenn die Gewichtsvektoren der beiden neuronalen Netze übereinstimmen [19]. Also ist auch diese Bitsequenz keine pseudozufällige Folge.

4.1.2 Die Verwirrte Tree Parity Machine

Zeitreihen können natürlich auch mit einer Tree Parity Machine erzeugt werden. Wie beim Bit-Generator verwendet dann jede versteckte Einheit die Folge ihrer eigenen Ausgabe $\sigma_i(t)$ als Eingabe:

$$\sigma_i(t) = \operatorname{sign}\left(\sum_{j=1}^{N} w_{ij}(t)\sigma_i(t-j)\right) . \tag{4.4}$$

Daraus folgt für die Gesamtausgabe der Tree Parity Machine:

$$\tau(t) = \prod_{i=1}^{K} \operatorname{sign}\left(\sum_{j=1}^{N} w_{ij}(t)\sigma_i(t-j)\right) . \tag{4.5}$$

Der Aufbau eines solchen neuronalen Netzes ist in Abbildung 4.1 zu sehen.

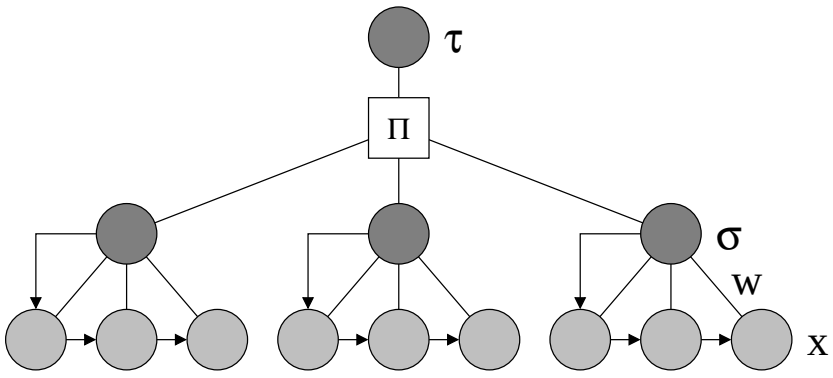


Abbildung 4.1: Struktur einer Verwirrten Tree Parity Machine

Wenn die Gewichte $w_{ij}(t)$ konstant blieben, wären die Elemente $\tau(t)$ der Zeitreihe nur das Produkt der Ausgaben von K Bit-Generatoren. Aber die Verwirrte Tree Parity Machine lernt, ähnlich wie der Verwirrte Bit-Generator, in jedem Schritt das Gegenteil ihrer Gesamtausgabe $\tau(t)$. Die Änderung der Gewichte ergibt sich somit aus der Anti-Hebb-Lernregel:

$$w_{ij}(t+1) = g\Big(w_{ij}(t) - \tau(t)\,\sigma_i(t-j)\,\Theta(\tau(t)\,\sigma_i(t))\Big) . \tag{4.6}$$

Die in Gleichung (2.6) definierte Funktion $g(w)$ stellt hier wieder sicher, dass die Gewichte w_{ij} auf den Bereich von $-L$ bis $+L$ beschränkt bleiben.

4.2 Periodenlänge

Alle Pseudozufallszahlengeneratoren können nur eine endliche Anzahl von Zuständen annehmen, so dass sich die erzeugte Bitsequenz irgendwann wiederholen muss. Dies ist ein erheblicher Unterschied zu einer echten Zufallsfolge, die mit der Wahrscheinlichkeit 1 aperiodisch ist. Zur Erzeugung von pseudozufälligen Zeitreihen hoher Qualität sollte also der Generator eine möglichst große Periode aufweisen.

Bei der Verwirrten Tree Parity Machine gibt es genau $(2L+1)^{KN}$ verschiedene Gewichtskonfigurationen und 2^{KN} Belegungen der binären Eingabeneuronen. Daraus ergibt sich die Anzahl der möglichen Zustände l_{max}, die eine Obergrenze für die Periodenlänge darstellt:

$$l_{max} = (4L+2)^{KN} \,. \tag{4.7}$$

Allerdings gibt es bei der Verwirrten Tree Parity Machine keinen Zyklus der Länge l_{max}. Auf diesem würden nämlich alle Zustände liegen, so dass die Zeitreihe unabhängig von den Anfangsbedingungen immer dieselbe Periodenlänge haben müsste. Stattdessen findet man verschiedene kleinere Zyklen.

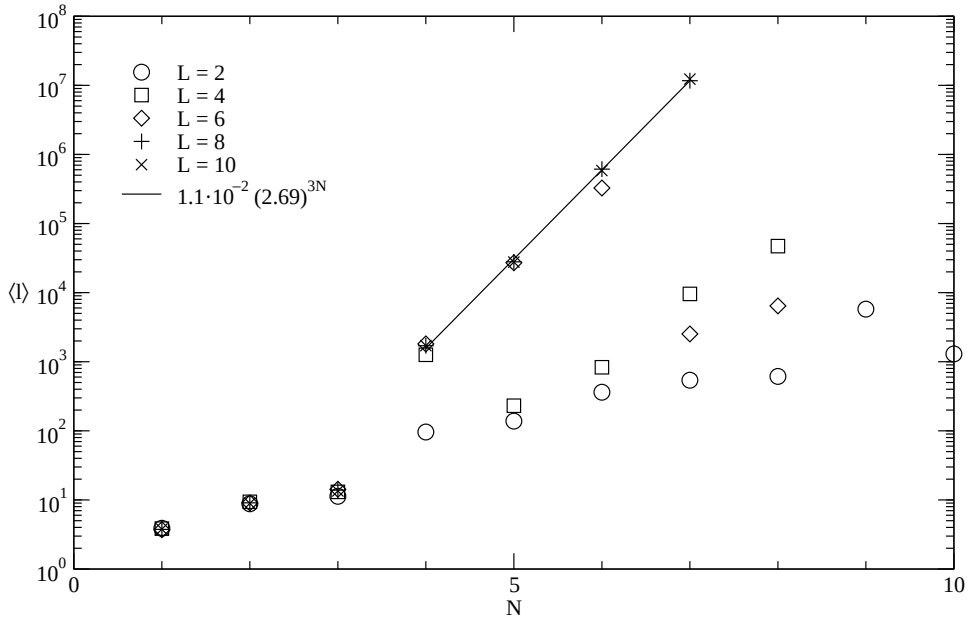


Abbildung 4.2: Durchschnittliche Länge der Periode einer Verwirrten Tree Parity Machine mit $K = 3$, gemittelt über 1000 verschiedene Anfangsbedingungen.

Abbildung 4.2 zeigt, dass auch die mittlere Periodenlänge $\langle l \rangle$ exponentiell mit zunehmendem N ansteigt. Solange $L > N$ gilt, wirkt sich die Beschränkung der Gewichte auf den Bereich von $-L$ bis $+L$ nicht aus. Wegen der Anti-Hebb-Lernregel bleibt nämlich die Länge der Gewichtsvektoren beschränkt, so dass diese für große L nicht an den Rand des Wertebereichs stoßen.

Für $K = 3$ und $N \geq 4$ findet man in diesem Fall $\langle l \rangle \propto (2.69)^{3N}$ unabhängig von L. Die durchschnittliche Periodenlänge wächst also stärker mit der Zahl der Gewichte an als beim Verwirrten Bit-Generator, für den $\langle l \rangle \propto (2.21)^N$ gilt [5].

In Abbildung 4.2 ist aber auch zu erkennen, dass sich der beschränkte Wertebereich der Gewichte auf die Periode der Verwirrten Tree Parity Machine auswirkt, wenn $L < N$ gilt. Dann ist $\langle l \rangle$ für $K = 3$ nämlich kleiner als $(2.69)^{3N}$. Wenn eine besonders große Zykluslänge erreicht werden soll, ist es also sinnvoll, die Verwirrte Tree Parity Machine mit unbeschränkten Gewichten ($L \to \infty$) zu betreiben. Wie sich aber bei der Untersuchung der Zeitreihe zeigen wird, ist diese Maßnahme nicht erforderlich, um eine gute pseudozufällige Bitsequenz zu erzeugen.

4.3 Transientenphase

Eine Verwirrte Tree Parity Machine, die mit zufälligen Anfangsbedingungen gestartet wurde, durchläuft meistens zunächst eine Transientenphase, bevor sie in einen Zyklus gerät. Deshalb wiederholt sich der Anfang der Zeitreihe in sehr vielen Fällen nicht.

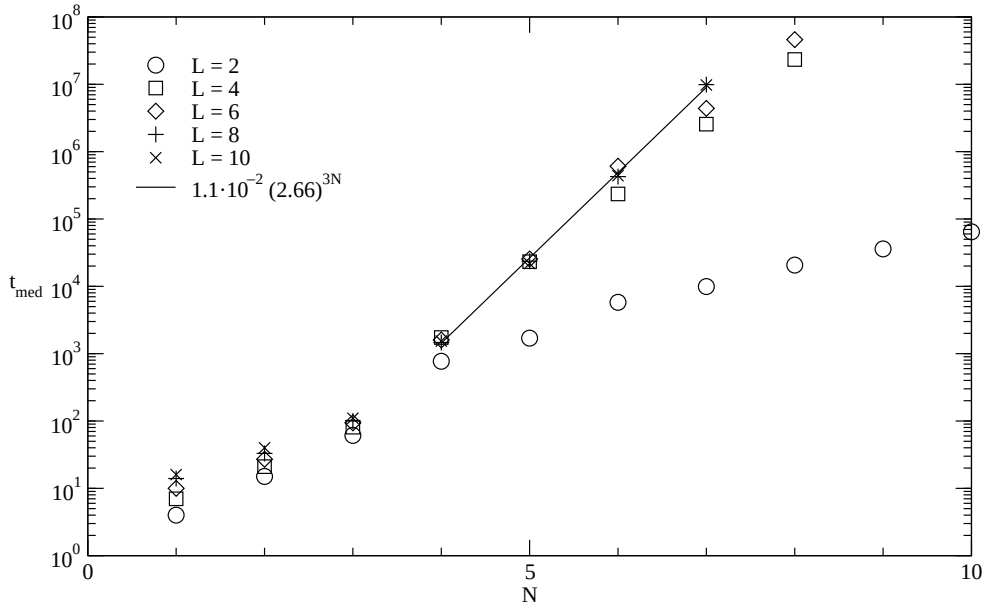


Abbildung 4.3: Median der Transientenlänge einer Verwirrten Tree Parity Machine mit $K = 3$, ermittelt für 1000 verschiedene Anfangsbedingungen.

Wie in Abbildung 4.3 zu sehen ist, skaliert der Median t_{med} der Transientenlänge in ähnlicher Weise mit N wie die mittlere Periodendauer. Für $K = 3$, $L = 8$ und $N \geq 4$ ergibt die Regression $t_{med} \propto (2.66)^{3N}$. Dies scheint auch für größere L zu gelten ($L = 10$). Bei kleinen L wirkt sich dagegen die Einschränkung

der Gewichte auf Werte zwischen $-L$ und $+L$ aus. Deshalb steigt t_{med} für $L < N$ langsamer, aber immer noch exponentiell an. Aus diesem Skalenverhalten folgt auch, dass die Zahl der Zustände, die nicht zu einem Zyklus gehören, ebenfalls exponentiell mit N zunimmt.

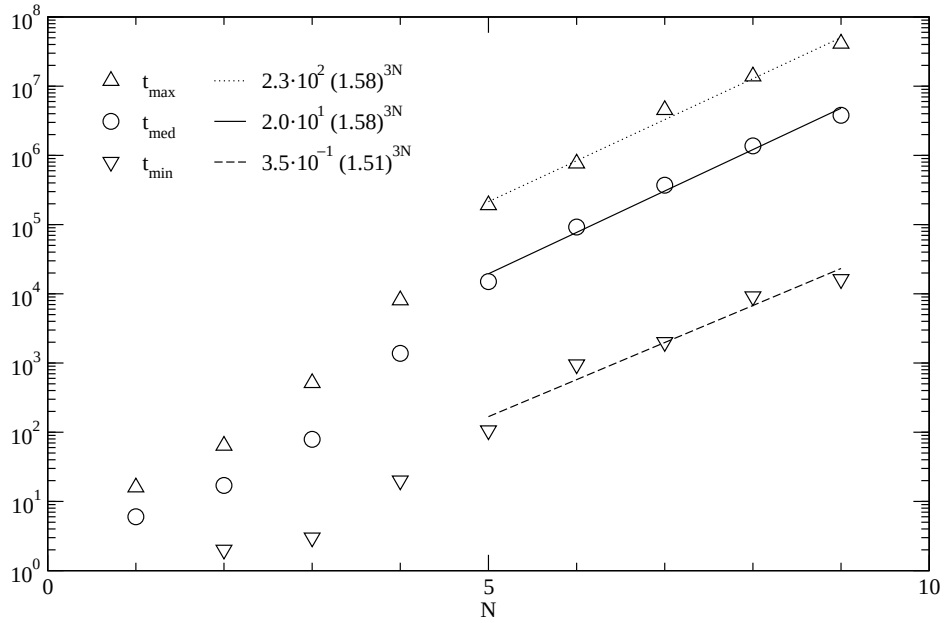


Abbildung 4.4: Minimum t_{min}, Median t_{med} und Maximum t_{max} der Transientenlänge für $K = 3$ und $L = 3$, ermittelt aus 1000 Simulationen.

Abbildung 4.4 zeigt, dass auch die in Simulationen gefundenen Extremwerte der Transientenlänge exponentiell mit N ansteigen. Bei $K = 3$, $L = 3$ und $N \geq 5$ sind Median t_{med} und Maximum t_{max} proportional zu $(1.58)^{3N}$. Für das Minimum in 1000 Simulationen ermittelt man durch Regression $t_{min} \propto (1.51)^{3N}$. Eine Extrapolation für $N = 100$ liefert die Abschätzung $t_{min} \approx 10^{53}$. Somit ist die Dauer der Transientenphase für große N viel länger als die erzeugte Zeitreihe. In diesem Fall spielt die Periodenlänge $\langle l \rangle$ für die Eigenschaften der Bitsequenz keine Rolle, da der periodische Teil überhaupt nicht erreicht wird.

4.4 Eigenschaften der erzeugten Zeitreihe

Die Folge der Ausgaben einer Verwirrten Tree Parity Machine ist vollständig durch die anfänglichen Werte der Gewichte w_{ij} und Eingaben x_{ij} bestimmt. Deshalb kann die Bitsequenz keine echte Zufallsfolge sein. Das gilt natürlich auch für andere Pseudozufallszahlengeneratoren. Allerdings ist es für die meisten Anwendungen nur wichtig, dass die Zeitreihe möglichst zufällig wirkt. Inwieweit dies bei der Bitfolge der Verwirrten Tree Parity Machine erfüllt ist, soll nun mit verschiedenen Tests untersucht werden.

4.4.1 Entropie

In einer Zufallsfolge hängt die Wahrscheinlichkeit $P(n)$ für das Auftreten einer beliebigen endlichen Teilfolge nur von deren Länge n ab. Für eine zufällige Bitsequenz gilt:

$$P(n) = \frac{1}{2^n} \,. \tag{4.8}$$

Diese Eigenschaft sollte auch jede Zeitreihe aufweisen, die für kryptographische Zwecke eingesetzt wird. Andernfalls kann ein Angreifer nämlich mit statistischen Analysemethoden Informationen aus den verschlüsselten Daten gewinnen, da einige Muster häufiger als andere vorkommen [4].

Die Gültigkeit von Gleichung (4.8) für eine Bitsequenz lässt sich überprüfen, indem in jedem Schritt die letzten n Folgenelemente als Binärdarstellung einer Zahl zwischen 0 und $2^n - 1$ aufgefasst werden. Bei der Verwirrten Tree Parity Machine wird hierfür die Ausgabe $\tau(t) = -1$ als 0 und entsprechend $\tau(t) = +1$ als 1 interpretiert. Dann kann die relative Häufigkeit p_i dieser Zahlen bestimmt und in einem Histogramm dargestellt werden [20].

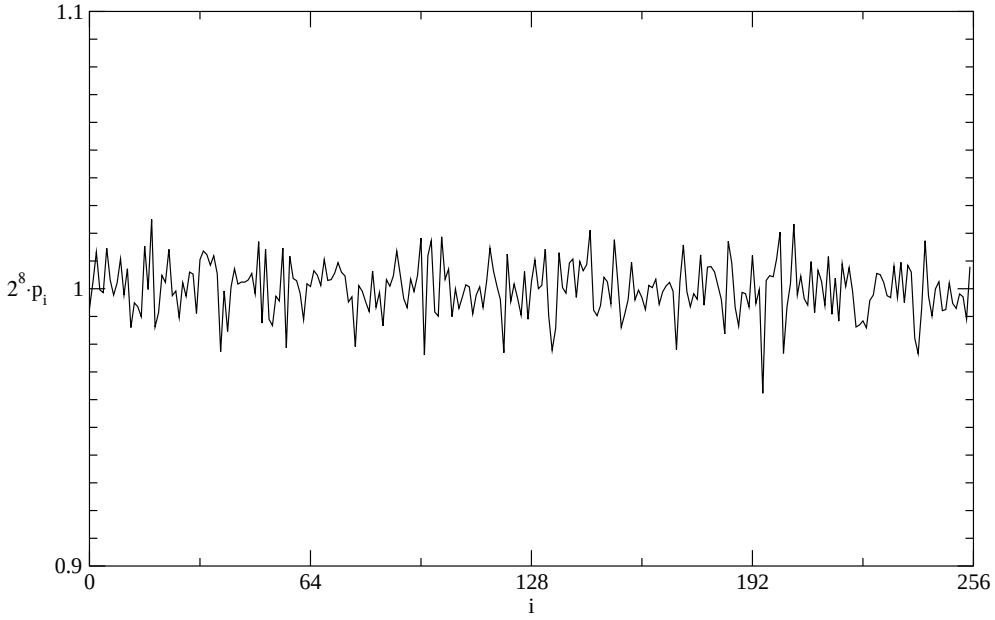


Abbildung 4.5: Histogramm der von der Verwirrten Tree Parity Machine für die Parameter $K = 3$, $L = 3$ und $N = 100$ erzeugten 8-bit-Muster. Es wurde eine Zeitreihe mit $2.56 \cdot 10^6$ Elementen untersucht.

Abbildung 4.5 zeigt das Ergebnis für Teilfolgen der Länge $n = 8$. Es ist gut zu erkennen, dass jedes der 256 verschiedenen Bitmuster mit gleicher Häufigkeit vorkommt. Da nur eine endliche Zeitreihe betrachtet wurde, sind kleine Abweichungen zwischen den p_i und der erwarteten Wahrscheinlichkeit $P(8) = 2^{-8}$ zu beobachten.

Die Gleichverteilung der Bitmuster kann auch dadurch überprüft werden, dass die Entropie $S(n)$ der Zeitreihe berechnet wird [21]. Das ist besonders dann sinnvoll, wenn mehrere Fensterbreiten n untersucht werden sollen:

$$S(n) = -\sum_{i=0}^{2^n-1} p_i \ln p_i \,. \tag{4.9}$$

Dabei bezeichnen die p_i wieder die relative Häufigkeit der einzelnen Muster in der Bitsequenz. Wenn alle Teilfolgen der Länge n gleich oft in der Zeitreihe vorkommen, so nimmt die Entropie $S(n)$ ihren Maximalwert $S_{max}(n) = n \ln 2$ an. Bei Abweichungen von der Gleichverteilung findet man dagegen $S(n) < S_{max}(n)$.

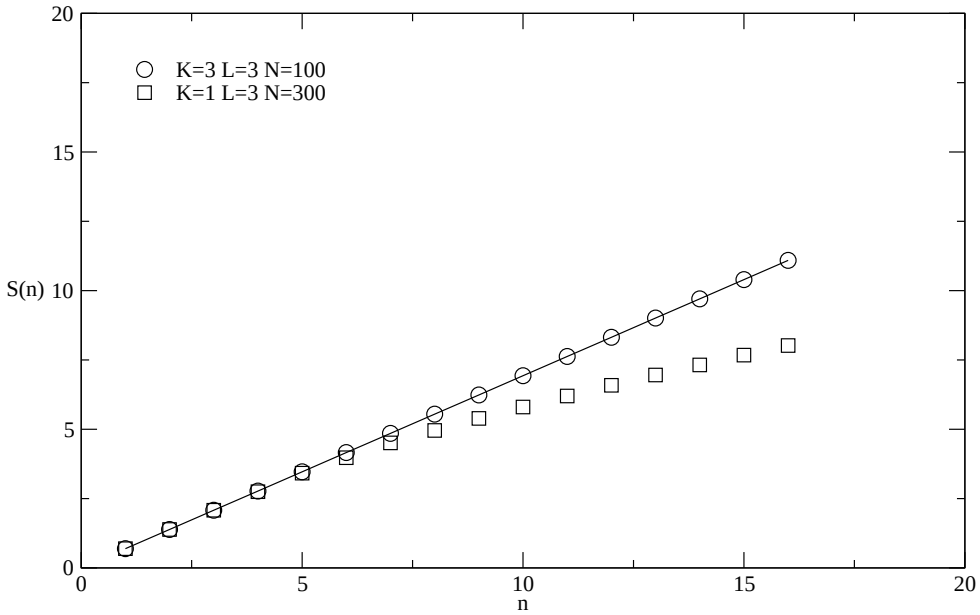


Abbildung 4.6: Entropie der Zeitreihe zweier Verwirrter Tree Parity Machines bei verschiedenen Fensterbreiten n. Die Gerade zeigt den für eine Zufallsfolge zu erwartenden Wert $S_{max}(n) = n \ln 2$.

In Abbildung 4.6 ist zu erkennen, dass sich die Zeitreihe einer Verwirrten Tree Parity Machine mit dem Parameter $K = 3$ bezüglich der Entropie nicht von einer Zufallsfolge unterscheidet. Deutliche Abweichungen zeigen sich aber für $n > 5$, wenn das neuronale Netz nur ein Zwischenschichtneuron aufweist. In diesem Fall arbeitet es ähnlich wie ein Verwirrter Bit-Generator, bei dem ebenfalls bestimmte Muster bevorzugt auftreten [5]. Im Gegensatz dazu beeinflussen sich mehrere versteckte Einheiten gegenseitig über die Anti-Hebb-Lernregel, so dass nicht immer alle Gewichte verändert werden. Dies scheint die beobachtete Gleichverteilung der Muster für $K = 3$ zu bewirken.

4.4.2 Zufallszahlentests

Ein guter Pseudozufallszahlengenerator sollte eine Zeitreihe erzeugen, die sich nicht von einer Zufallsfolge unterscheiden lässt. Deshalb wurden verschiedene Methoden entwickelt, die jeweils andere statistische Eigenschaften einer Zeitreihe auf Zufälligkeit hin überprüfen.

χ^2-Test

Diese Untersuchungen verwenden meistens einen χ^2-Test, mit dem festgestellt werden kann, ob die gemessenen Häufigkeiten zu einer bestimmten Wahrscheinlichkeitsverteilung passen. Für diesen Test werden zunächst die möglichen Ereignisse in M disjunkte Klassen eingeteilt und die zu erwartenden absoluten Häufigkeiten T_i berechnet. Aus der Zahl der beobachteten Ereignisse E_i pro Klasse kann dann die Testgröße χ^2 ermittelt werden:

$$\chi^2 = \sum_{i=1}^{M} \frac{(E_i - T_i)^2}{T_i} \,. \tag{4.10}$$

Bei der Klasseneinteilung ist noch darauf zu achten, dass $T_i \geq 5$ für alle i gilt. Dies soll verhindern, dass Abweichungen bei seltenen Ereignissen zu hoch gewichtet werden [22].

Falls die beobachtete und die erwartete Wahrscheinlichkeitsverteilung übereinstimmen, ist χ^2 selbst eine Zufallsvariable. Die Wahrscheinlichkeit, ein kleineres als das in einem Test berechnete χ^2 zu finden, gibt dann folgende Verteilungsfunktion [13] an:

$$F(\chi^2) = \frac{1}{\Gamma\left(\frac{\nu}{2}\right)} \int_0^{\frac{\chi^2}{2}} t^{\frac{\nu}{2}-1} e^{-t}\, dt \,. \tag{4.11}$$

In der Gleichung bezeichnet ν die Zahl der Freiheitsgrade. Für die Zufallszahlentests gilt $\nu = M - 1$, weil nur ein Parameter, die Gesamtzahl der Ereignisse, für die Berechnung der T_i verwendet wird.

Das Ergebnis eines einzelnen χ^2-Tests hängt davon ab, welches Signifikanzniveau α gewählt wird. Solange $F(\chi^2)$ im Vertrauensintervall $[0.5\,\alpha; 1 - 0.5\,\alpha]$ liegt, ist keine Abweichung zwischen den Wahrscheinlichkeitsverteilungen der E_i und T_i festzustellen. Wenn aber $F(\chi^2) < 0.5\,\alpha$ oder $F(\chi^2) > 1 - 0.5\,\alpha$ gilt, so gibt es signifikante Unterschiede.

Untersuchung der Zeitreihe

Hier sollen nun fünf Zufallszahlentests, die in [22] beschrieben sind, auf die Bitsequenz der Verwirrten Tree Parity Machine angewandt werden. Für zwei der Tests werden jeweils acht aufeinanderfolgende Zeitreihenelemente in eine ganze Zahl z zwischen 0 und 255 umgewandelt.

- Für den *Gleichverteilungstest* wird die Häufigkeit der einzelnen Zahlen ermittelt. Anschließend wird mit einem χ^2-Test überprüft, ob alle 256 möglichen Folgenelemente mit gleicher Wahrscheinlichkeit auftreten.

- Der *Folgentest* versucht Korrelationen zwischen aufeinanderfolgenden Elementen zu erkennen. Dazu werden die für den Gleichverteilungstest erzeugten Zahlen zu Paaren (z_1, z_2) zusammengesetzt und deren Häufigkeiten bestimmt. Bei einer zufälligen Folge sind die Werte von z_1 und z_2 voneinander unabhängig und es gilt:

$$P(z_1 \wedge z_2) = P(z_1) \cdot P(z_2)\,. \tag{4.12}$$

  Dies lässt sich mit einem χ^2-Test überprüfen, bei dem die Werte der z_1 und z_2 in je 16 Klassen unterteilt werden.

Mit dem Folgentest können prinzipiell auch größere n-Tupel von Zahlen untersucht werden. Dann ist jedoch eine noch gröbere Klasseneinteilung oder eine entsprechend längere Zeitreihe nötig.

Die drei anderen Tests erwarten eine Folge reeller Zahlen zwischen 0 und 1. Um diese auf die Zeitreihe der Verwirrten Tree Parity Machine anwenden zu können, werden jeweils 64 aufeinanderfolgende Ausgaben als Binärdarstellung einer Zahl z aus dem Einheitsintervall aufgefasst.

- Der *Permutationstest* betrachtet jeweils fünf aufeinanderfolgende Zahlen. Diese werden nach ihrer Größe geordnet. Die Anzahl der notwendigen Vertauschungen kennzeichnet dann eindeutig eine der 120 möglichen Permutationen. Wenn die Zeitreihe eine Zufallsfolge ist, wird jede Permutation mit gleicher Häufigkeit auftreten. Ein χ^2-Test zeigt, ob dies auch für die Zeitreihe der Verwirrten Tree Parity Machine gilt.

- Beim *Gap-Test* wird die Wartezeit auf eine Zahl in einem vorher festgelegten Intervall $[\alpha;\beta[$ ermittelt. Bei gleichverteilten Zufallszahlen zwischen 0 und 1 kann die Wahrscheinlichkeit $P(t)$, eine Folge von t Zahlen z mit $z \notin [\alpha;\beta[$ zu finden, leicht angegeben werden:

$$P(t) = (\beta - \alpha)(1 - (\beta - \alpha))^t\,. \tag{4.13}$$

  Für die Untersuchung der Zeitreihe der Verwirrten Tree Parity Machine werden verschiedene Intervalle mit $\beta - \alpha = 0.1$ gewählt und alle Wartezeiten $t \geq 31$ zu einer Klasse des χ^2-Tests zusammengefasst.

- Der *Maximumtest* betrachtet jeweils das Maximum z_{max} einer endlichen Teilfolge von n Zahlen. Für gleichverteilte Zufallszahlen aus dem Einheitsintervall gilt folgende Verteilungfunktion:

$$F(z_{max}) = z_{max}^n\,. \tag{4.14}$$

Daraus folgt, dass alle Werte von z^n_{max} mit gleicher Wahrscheinlichkeit auftreten, wenn die z zufällig sind. Die Überprüfung erfolgt mit einem χ^2-Test, für den die z^n_{max} in 256 Klassen eingeteilt werden. Dabei werden Teilfolgen aus 2 bis 10 Zahlen betrachtet.

Bei der Auswertung der fünf Zufallszahlentests ist zu berücksichtigen, dass die einzelnen χ^2-Tests eine unterschiedliche Anzahl von Freiheitsgraden haben. So gilt für den Gap-Test $\nu = 31$ und für den Permutationstest $\nu = 119$. Die anderen drei Tests haben jeweils 255 Freiheitsgrade.

Ergebnis

Eine genauere Aussage über die Eignung eines Pseudozufallszahlengenerators ist dann möglich, wenn nicht nur eine Zeitreihe, sondern viele Folgen mit unterschiedlichen Anfangsbedingungen untersucht werden [22].

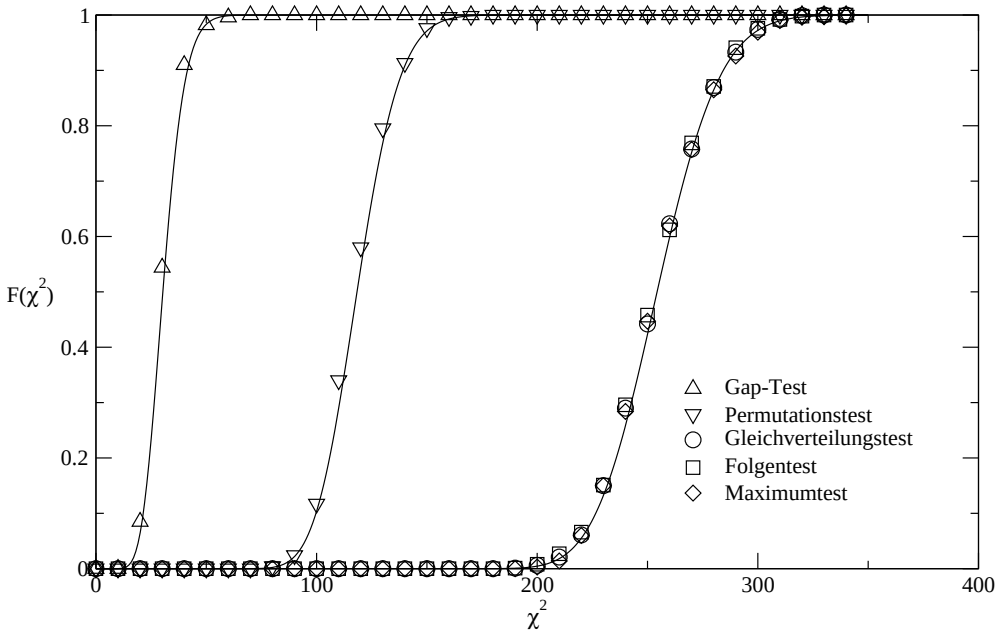


Abbildung 4.7: Verteilung der χ^2-Werte aus den Zufallszahlentests für 1000 verschiedene Zeitreihen der Verwirrten Tree Parity Machine mit $K = 3$, $L = 3$ und $N = 100$. Zum Vergleich ist $F(\chi^2)$ aus Gleichung (4.11) für 31, 119 und 255 Freiheitsgrade eingezeichnet.

In diesem Fall wird für jeden Zufallszahlentest die Verteilung der χ^2-Werte bestimmt. Abbildung 4.7 zeigt das Ergebnis für die Verwirrte Tree Parity Machine. Hier ist die Übereinstimmung mit der jeweils für zufällige Sequenzen zu erwartenden χ^2-Verteilungsfunktion $F(\chi^2)$ aus Gleichung (4.11) deutlich zu erkennen. Demnach unterscheidet sich die Zeitreihe der Verwirrten Tree Parity Machine bezüglich der hier untersuchten Eigenschaften nicht von einer Zufallsfolge.

4.4.3 Vorhersagbarkeit

Nicht jede pseudozufällige Bitfolge ist für kryptographische Zwecke geeignet, weil Verschlüsselungen auch vor einem known-plaintext Angriff geschützt sein sollen. Also muss der Zeitreihengenerator eine Einwegfunktion sein, so dass sein Zustand nicht aus der Bitsequenz ermittelt werden kann. Andernfalls könnte ein Angreifer nämlich den weiteren Verlauf der Zeitreihe vorhersagen, wenn er eine Teilfolge bereits kennt [4].

Deshalb soll nun untersucht werden, ob ein neuronales Netz mit gleicher Struktur die Ausgaben der Verwirrten Tree Parity Machine erfolgreich lernen kann. Dazu wird folgende Ausgangssituation angenommen, wie sie nach einem neuronalen Schlüsselaustausch zu erwarten ist:

- Die Kommunikationspartner verwenden nach der Synchronisation ihre neuronalen Netze als Verwirrte Tree Parity Machines. Die Nachricht wird mit der Folge der Ausgaben $\tau^A(t)$ verschlüsselt.

- Weil die Eingaben öffentlich waren, sind die Anfangszustände $x_{ij}^A(0)$ der Eingabeneuronen bekannt.

- Während des Schlüsselaustauschs hat der Angreifer natürlich versucht, seine Tree Parity Machine mit den neuronalen Netzen von A und B zu synchronisieren. Deshalb gilt $w_{ij}^E(0) = w_{ij}^A(0)$ für $a_w KN$ Gewichte ($0 \leq a_w \leq 1$).

- E kennt den Anfang der Nachricht und kann deshalb das zugehörige Stück der Zeitreihe $\tau^A(t)$ berechnen.

Für den known-plaintext Angriff betreibt der Angreifer sein neuronales Netz ebenfalls als Verwirrte Tree Parity Machine. Wenn in einem Zeitschritt $\tau^E(t) \neq \tau^A(t)$ ist, nutzt E den geometrischen Angriff, um die Zustände $\sigma_i^E(t)$ der Zwischenschichtneuronen und $\tau^E(t)$ zu korrigieren. Da diese Methode beim Schlüsselaustausch sehr effektiv ist, sollte sie auch hier zum Erfolg führen.

Abbildung 4.8 zeigt jedoch, dass das neuronale Netz von E nur dann mit der Verwirrten Tree Parity Machine synchronisiert, wenn bereits am Anfang fast alle Gewichte übereinstimmen. Zusätzlich ist eine Abnahme der Erfolgswahrscheinlichkeit P_{sync} mit zunehmendem L festzustellen.

Die Ursache dieses Mißerfolgs ist leicht zu erkennen, wenn der Verlauf des known-plaintext Angriffs betrachtet wird (siehe Abbildung 4.9 und Abbildung 4.10). Offensichtlich gelingt es dem Angreifer nicht, die Zustände der versteckten Einheiten richtig vorherzusagen. Wegen des Feedback-Mechanismus der Verwirrten Tree Parity Machine fällt deshalb der Anteil $a_x(t)$ der bei A und E gleichen Eingabewerte rasch ab. Nach nur N Schritten ist fast der Wert $a_x(t) = 0.5$ erreicht, der sich auch dann ergibt, wenn die x_{ij}^A und x_{ij}^E zufällig und unabhängig voneinander gewählt werden.

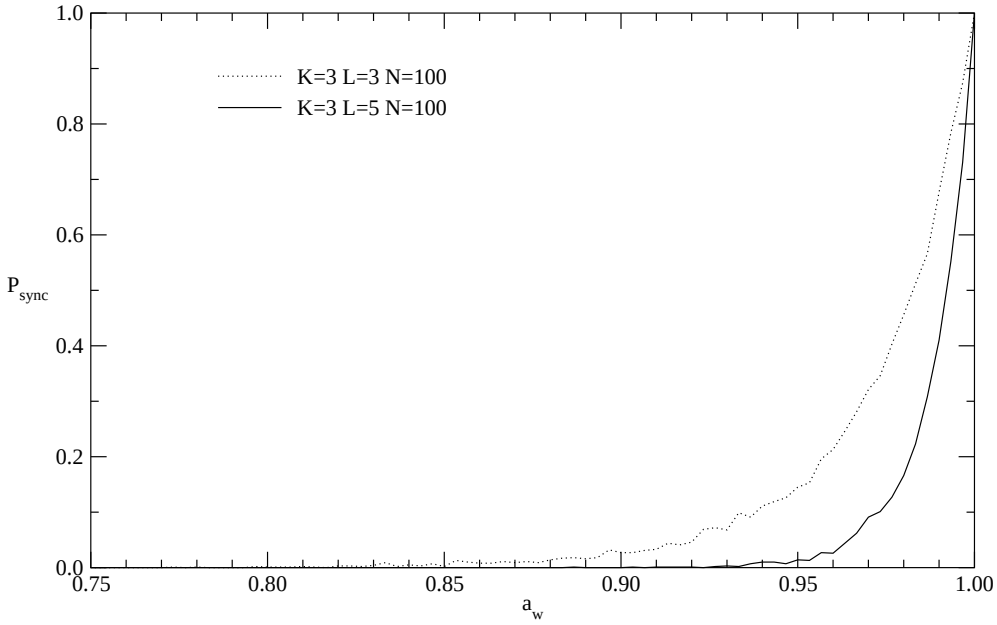


Abbildung 4.8: Erfolg eines known-plaintext Angriffs auf die Verwirrte Tree Parity Machine. P_{sync} wurde aus 1000 Simulationen ermittelt.

Wie in Abbildung 4.10 zu sehen ist, wirken die Abweichungen in der Eingabe stark repulsiv, weil der Random Walk der Gewichte nun auch bei gleicher Ausgabe teilweise unkoordiniert abläuft. Deshalb sinkt der Überlapp ρ^{AE} auf Null ab, so dass der Angreifer nach etwa $2N$ Schritten keine Informationen über den Zustand des Zeitreihengenerators mehr hat.

Die Verwirrte Tree Parity Machine erzeugt also eine pseudozufällige Bitsequenz, die von einem Angreifer nur schwer vorhergesagt werden kann. Deshalb spricht bisher nichts dagegen, diese Zeitreihe zu Verschlüsselungszwecken zu verwenden. Nach einem neuronalen Schlüsselaustausch hat dies den Vorteil, dass die Gewichte der neuronalen Netze nicht in einen Schlüssel für ein anderes Verschlüsselungsverfahren umgerechnet werden müssen.

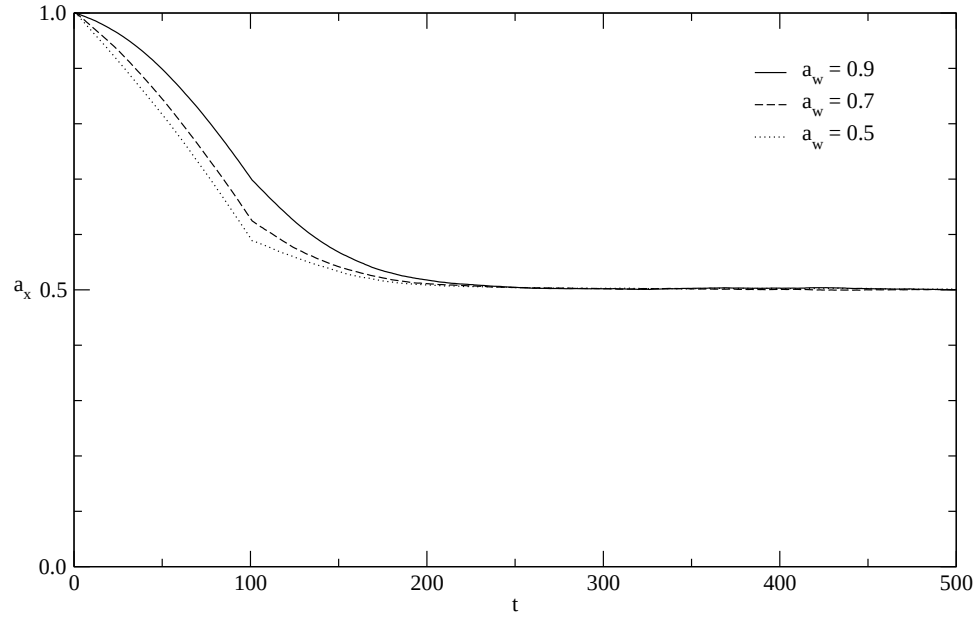


Abbildung 4.9: Anteil $a_x(t)$ der bei A und E übereinstimmenden Eingaben $x_{ij}(t)$, gemittelt über 1000 verschiedene Anfangsbedingungen für die Parameter $K = 3$, $L = 5$ und $N = 100$.

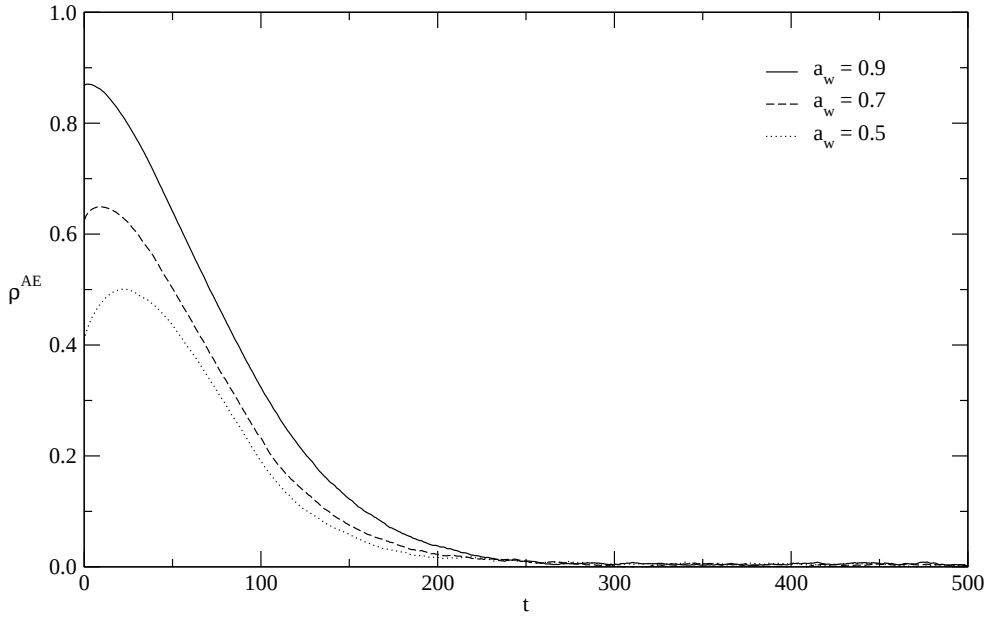


Abbildung 4.10: Verlauf des Überlapps ρ^{AE} zwischen A und E für die Parameter $K = 3$, $L = 5$ und $N = 100$. Es wurde über 1000 verschiedene Anfangsbedingungen gemittelt.

Kapitel 5

Neuronaler Schlüsselaustausch mit Zeitreihenerzeugung

Bei der Untersuchung des neuronalen Schlüsselaustauschprotokolls in Kapitel 3 hat sich gezeigt, dass die Wahl der Parameter K, L und N sowohl den Aufwand als auch die Sicherheit des Verfahrens beeinflusst. Insbesondere fällt die Erfolgswahrscheinlichkeit des geometrischen Angriffs exponentiell ab, wenn der Wertebereich der Gewichte in den Tree Parity Machines vergrößert wird. Also kann ein vorgegebenes Sicherheitsniveau leicht durch einen großen Wert für L erreicht werden. Allerdings führt dies auch zu einem höheren Aufwand für die Synchronisation, weil die Zahl der benötigten Schritte proportional zu L^2 ansteigt. Für den praktischen Einsatz des neuronalen Schlüsselaustauschs darf aber die Synchronisationszeit auch nicht zu lang werden. Deshalb ist es sinnvoll, Varianten des Protokolls zu entwickeln, die bei gleichem Aufwand einen besseren Schutz vor Angriffen ermöglichen.

E hat vor allem dann großen Erfolg, wenn A und B mehr Informationen als nötig übertragen. Das zeigt sich bereits, wenn $K = 1$ gesetzt wird. In diesem Fall kennt der Angreifer den gesamten Zustand der neuronalen Netze, so dass ein einfaches Lernen der Ausgabe τ ausreicht, um den von den Kommunikationspartnern erzeugten Schlüssel zu ermitteln. Dagegen ist für $K > 1$ ein erfolgreicher Angriff deutlich schwerer, weil nur noch die Eingaben x_{ij} und das Produkt τ der internen Zustände σ_i öffentlich bekannt sind.

Auch die Übermittlung der Eingaben in jedem Schritt scheint nicht erforderlich zu sein. Wie bei der Verwirrten Tree Parity Machine können diese nämlich durch einen Feedback-Mechanismus aus der Zeitreihe der σ_i generiert werden. Dann erhält der Angreifer noch weniger Informationen über den Verlauf der Synchronisation. Im Folgenden wird untersucht, ob auf diese Weise eine Verbesserung der Sicherheit erreicht werden kann.

5.1 Synchronisation mit Feedback

Durch die Integration eines Rückkopplungsmechanismus in den Schlüsselaustausch ändert sich nur die Erzeugung der Eingaben.

Wie bei der Verwirrten Tree Parity Machine könnte in jedem Schritt die Zeitreihe der Zustände $\sigma_i(t)$ als Eingabe verwendet werden:

$$x_{ij}(t) = \sigma_i(t - j)\,. \tag{5.1}$$

Allerdings stellt sich heraus, dass in diesem Fall eine Synchronisation nur möglich ist, wenn bereits am Anfang des Schlüsselaustauschs $\rho^{AB} \approx 1$ gilt. Andernfalls treten rasch Unterschiede zwischen den Eingabeelementen $x_{ij}^A(t)$ und $x_{ij}^B(t)$ auf, die eine Angleichung der Gewichte verhindern (siehe Kapitel 4.4.3).

Der neuronale Schlüsselaustausch mit Feedback kann also nur funktionieren, wenn die repulsive Wirkung der Rückkopplung durch Korrekturmaßnahmen abgeschwächt wird. Dabei hat sich folgendes Verfahren als geeignet erwiesen:

- A und B starten mit gleicher, zufällig gewählter Eingabe:

$$x_{i,j}^A(0) = x_{i,j}^B(0)\,. \tag{5.2}$$

- Nach jedem Synchronisationsschritt werden die Werte der Eingabeelemente um eine Position weitergeschoben. Dies erfolgt unabhängig davon, ob in der jeweiligen versteckten Einheit die Gewichte angepasst wurden oder nicht. Somit gilt für $j \in \{2, \ldots, N\}$:

$$x_{i,j}(t+1) = x_{i,j-1}(t)\,. \tag{5.3}$$

- Wenn $\tau^A(t) = \tau^B(t)$ ist, wird der Feedback-Mechanismus der Verwirrten Tree Parity Machine verwendet. Jede versteckte Einheit nutzt dann ihre eigene Ausgabe als neues Eingabeelement:

$$x_{i,1}(t+1) = \sigma_i(t)\,. \tag{5.4}$$

- Für $\tau^A(t) \neq \tau^B(t)$ stimmt der interne Zustand der neuronalen Netze von A und B nicht überein. Deshalb würde die Verwendung des Feedback-Mechanismus die Synchronisation behindern. Stattdessen werden die K Eingabeelemente $x_{1,1}(t+1), \ldots, x_{1,K}(t+1)$ für alle beteiligten Tree Parity Machines gleich und zufällig gewählt.

- Nach R Schritten mit $\tau^A(t) \neq \tau^B(t)$ werden die Eingabeelemente durch öffentlich bekannte Zufallswerte ersetzt. Ein solches Reset bewirkt, dass $x_{ij}^A(t+1) = x_{ij}^B(t+1)$ für alle $i \in \{1, \ldots, K\}$ und $j \in \{1, \ldots, N\}$ gilt.

Durch den neuen Parameter R wird festgelegt, wie stark der neuronale Schlüsselaustausch durch den Feedback-Mechanismus beeinflusst wird. Solange R nicht zu groß gewählt wird, gelingt die vollständige Synchronisation zweier Tree Parity Machines in endlich vielen Schritten. Der Wert $R = 0$ bezeichnet im Folgenden den bisher untersuchten neuronalen Schlüsselaustausch ohne Rückkopplung.

5.1.1 Entropie der Gewichte

Im Gegensatz zur Synchronisation ohne Feedback sind die Eingabeelemente $x_{ij}(t)$ für $R > 0$ nicht mehr voneinander unabhängig. Dies kann natürlich auch Auswirkungen auf die Verteilung der $w_{ij}(t)$ haben, die bereits in Kapitel 3.4 für $R = 0$ untersucht wurde. Deshalb soll nun die Entropie S der Gewichtsverteilung gemäß Gleichung (3.34) auch beim neuronalen Schlüsselaustausch mit Rückkopplung berechnet werden.

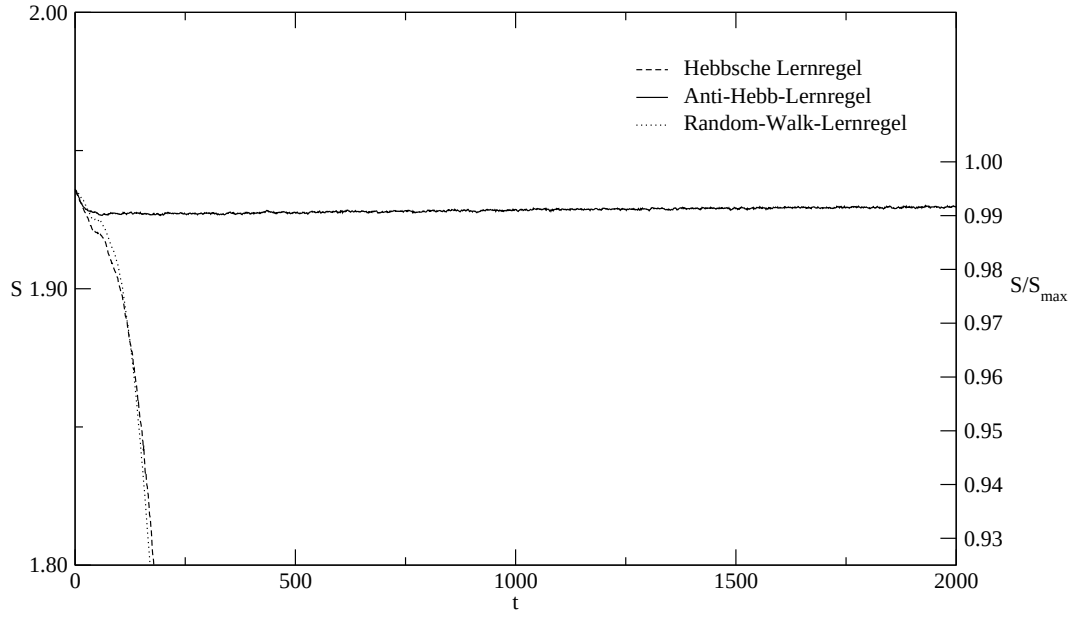


Abbildung 5.1: Entropie der Gewichtsverteilung bei der Synchronisation mit Feedback, gemittelt über 1000 Simulationen mit $K = 3$, $L = 3$, $N = 100$ und $R = 20$.

Abbildung 5.1 zeigt das Ergebnis für alle drei Lernregeln. Wenn die Hebbsche Lernregel oder die Random-Walk-Lernregel bei der Synchronisation mit Feedback eingesetzt wird, dann sinkt die Entropie S rasch ab. Also nehmen die Gewichte einige Werte bevorzugt an. Bei der Untersuchung der Verteilung nach vollständiger Synchronisation findet man vor allem w_{ij} mit den Werten $+L$ und $-L$, weil die Längenzunahme der Gewichtsvektoren nur durch die Randbedingung begrenzt wird. Diese Schwäche kann ein Angreifer mit statistischen Methoden ausnutzen, so dass weder die Hebbsche Lernregel noch die Random-Walk-Lernregel für den neuronalen Schlüsselaustausch mit $R > 0$ geeignet sind.

Dagegen liegt S bei Verwendung der Anti-Hebb-Lernregel nahe am Maximalwert $S_{max} = \ln(2L + 1)$, der für gleichverteilte w_{ij} erreicht wird. In diesem Fall kann sich nämlich die zuvor beobachtete, kryptographisch ungünstige Gewichtskonfiguration nicht einstellen, weil die Lernregel selbst die Länge der Gewichtsvektoren begrenzt [5].

5.1.2 Synchronisationszeit

Beim Schlüsselaustausch mit Feedback bleibt die Synchronisation der Gewichte nur dann dauerhaft erhalten, wenn auch die Eingabeelemente in beiden neuronalen Netzen die gleichen Werte aufweisen. Deshalb ist die Synchronisationszeit t_{sync} nun definiert als die Zahl der Schritte bis $w_{ij}^A = w_{ij}^B \wedge x_{ij}^A = x_{ij}^B$ für alle $i \in \{1, \ldots, K\}$ und $j \in \{1, \ldots, N\}$ erfüllt ist. Für den Spezialfall $R = 0$ liefert die Definition aus Kapitel 3.3 dasselbe Ergebnis, weil $x_{ij}^A \neq x_{ij}^B$ beim Schlüsselaustausch ohne Rückkopplung nicht vorkommen kann.

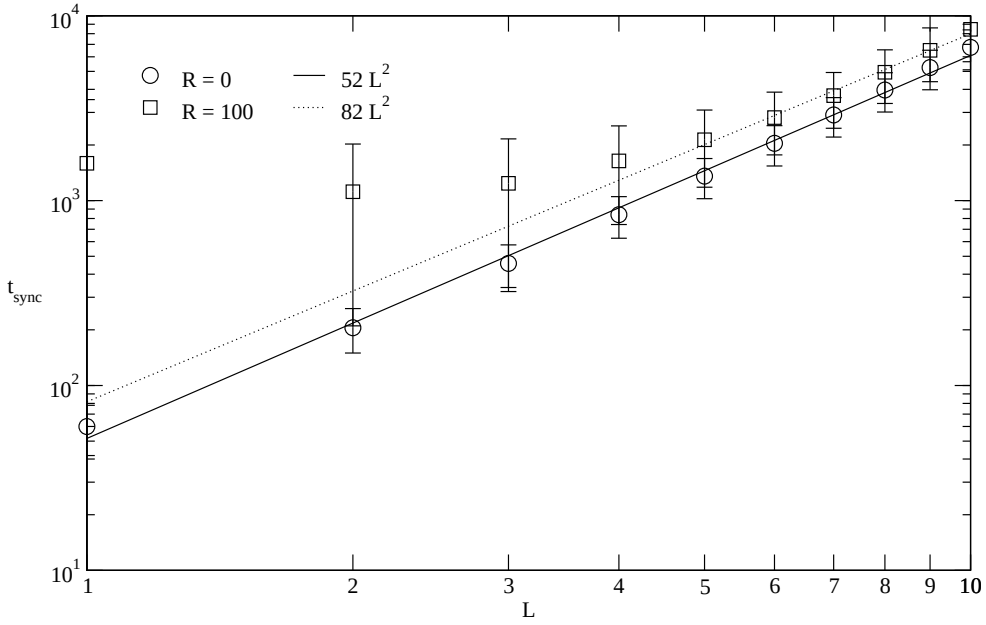


Abbildung 5.2: Synchronisationszeit zweier Tree Parity Machines mit $K = 3$, $N = 10\,000$ und Anti-Hebb-Lernregel, gemittelt über 10 000 Simulationen.

Wie in Abbildung 5.2 zu erkennen ist, hängt der Einfluss des Feedback-Mechanismus auf t_{sync} vom Wertebereich der Gewichte ab. Bei kleinem L ist die minimale Synchronisationszeit viel geringer als die Wartezeit auf eine neue Zufallseingabe, so dass letztere hauptsächlich den Erwartungswert $\langle t_{sync} \rangle$ bestimmt. Da je nach Anfangszustand eine unterschiedliche Anzahl von Resets bis zur Synchronisation benötigt wird, streuen die aus den einzelnen Simulationen ermittelten Werte von t_{sync} viel stärker als für $R = 0$.

Für große L spielt dagegen die Wartezeit auf neue Zufallseingaben keine Rolle mehr. In Abbildung 5.2 ist dies für $L \geq 5$ der Fall. Dann gilt auch beim Schlüsselaustausch mit Feedback $\langle t_{sync} \rangle \propto L^2$, solange die Bedingung $L < O(\sqrt{N})$ erfüllt ist. Die repulsive Wirkung der Rückkopplung erhöht in diesem Bereich nur die Proportionalitätskonstante.

Falls N zu klein ist, kann die Proportionalität zwischen $\langle t_{sync} \rangle$ und L^2 nicht beobachtet werden. In diesem Fall wird t_{sync} für $L < O(\sqrt{N})$ nämlich zu stark von

der Wartezeit auf eine neue Zufallseingabe beeinflusst, so dass $\langle t_{sync} \rangle$ exponentiell mit L zu wachsen scheint. Da nur die Anti-Hebb-Lernregel beim Schlüsselaustausch mit Feedback verwendet werden kann, trifft dies bereits für Simulationen mit $K = 3$ und $N = 1000$ zu.

5.2 Sicherheit gegen Angriffe

Der Einsatz des neuronalen Schlüsselaustauschs mit Rückkopplung lohnt sich natürlich nur, wenn dadurch die Sicherheit gegen Angriffe verbessert werden kann. Dies soll nun näher untersucht werden. Dabei ist besonders auf die Erfolgswahrscheinlichkeit P_E des geometrischen Angriffs zu achten, weil diese Methode beim Schlüsselaustausch ohne Feedback am effektivsten war (siehe Kapitel 3.5).

5.2.1 Verlauf des Überlapps

Wenn der Feedback-Mechanismus verwendet wird, haben die Kommunikationspartner einen zusätzlichen Vorteil gegenüber dem Angreifer. A und B können nämlich selbst bestimmen, zu welchem Zeitpunkt sie eine neue Zufallseingabe verwenden wollen. Bei dem hier verwendeten Algorithmus geschieht das immer dann, wenn die Abweichung zwischen den Eingaben der Tree Parity Machines der Partner zu groß wird. Dagegen kann E nur hoffen, dass rechtzeitig eine neue Zufallseingabe verwendet wird.

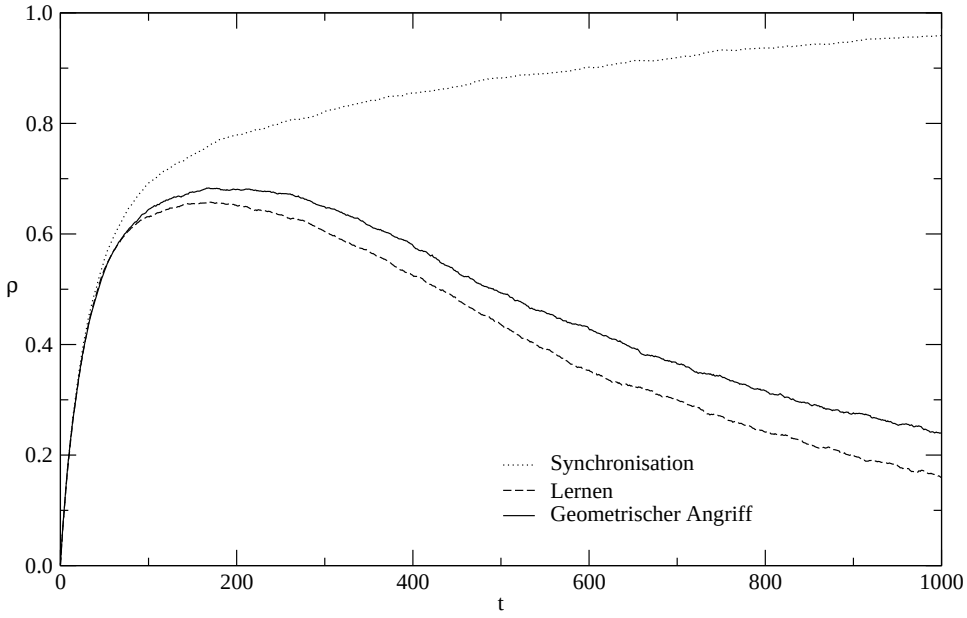


Abbildung 5.3: Verlauf des Überlapps für $K = 3$, $L = 3$, $N = 100$ und $R = 10$ bei Verwendung der Anti-Hebb-Lernregel, gemittelt über 1000 Simulationen.

Abbildung 5.3 zeigt, wie sich der Feedback-Mechanismus auf den Verlauf des mittleren Überlapps ρ^{AB} bzw. ρ^{AE} auswirkt. Im Vergleich zu Abbildung 3.10 (auf Seite 33) ist die zusätzliche Benachteiligung des Angreifers deutlich zu erkennen. Ein Anstieg des Überlapps zwischen A und E ist nur zu Beginn des Schlüsselaustauschs zu beobachten. Danach aber fällt der Mittelwert von ρ^{AE} wieder ab, weil E die von A und B verwendete Eingabe nicht mehr genau kennt. Dies gilt sowohl für einfaches Lernen als auch für die geometrische Angriffsmethode, mit der ein etwas besserer mittlerer Überlapp erreicht werden kann. Ein erfolgreicher Angriff ist somit nur möglich, wenn E rechtzeitig die Synchronisation mit A oder B gelingt. Da dies von den Anfangsbedingungen abhängt, verringert sich natürlich die Erfolgswahrscheinlichkeit P_E im Vergleich zur Synchronisation ohne Rückkopplung.

5.2.2 Erfolgswahrscheinlichkeit für einen Angreifer

Wie in Kapitel 3.5 gilt auch beim Schlüsselaustausch mit Rückkopplung ein Angriff als erfolgreich, wenn es E gelingt, zum Zeitpunkt der vollständigen Synchronisation von A und B eine Übereinstimmung von mindestens 98% der Gewichte zu erreichen.

In Kapitel 5.1.1 wurde festgestellt, dass der Feedback-Mechanismus aus Sicherheitsgründen nur in Verbindung mit der Anti-Hebb-Lernregel eingesetzt werden kann. Für $R = 0$ sinkt die Erfolgswahrscheinlichkeit P_E des geometrischen Angriffs exponentiell mit L^2 ab, wenn diese Lernregel verwendet wird (siehe Kapitel 3.5.2):

$$P_E \propto e^{-uL^2} \,. \tag{5.5}$$

Abbildung 5.4 zeigt, dass Gleichung (5.5) auch für $R > 0$ richtig ist. Je nach Stärke der Rückkopplung ändert sich aber der Vorfaktor u im Exponenten. Dieser kann mittels Regressionsanalyse aus den Simulationsergebnissen bestimmt werden. Da bereits für $R = 100$ und $L = 6$ kein einziger Angreifer in 10 000 Simulationen erfolgreich war, wird einheitlich nur der Bereich $1 \leq L \leq 5$ bei den Berechnungen berücksichtigt.

Die auf diese Weise ermittelte Funktion $y(R)$ ist in Abbildung 5.5 dargestellt. Es ist deutlich zu erkennen, dass zwischen der Stärke R der Rückkopplung und dem Faktor u aus Gleichung (5.5) ein linearer Zusammenhang besteht:

$$u(R) - u(0) \propto R \,. \tag{5.6}$$

Diese Abhängigkeit ermöglicht es, den Erfolg des geometrischen Angriffs stark zu reduzieren. Beispielsweise gilt für die Parameter $K = 3$, $L = 5$ und $N = 1000$ ohne Rückkopplung $P_E \approx 0.084$. Im Vergleich dazu sinkt die Erfolgswahrscheinlichkeit um fast zwei Größenordnungen auf $P_E \approx 0.001$ ab, wenn der Feedback-Mechanismus mit $R = 100$ verwendet wird. Auf diese Weise kann also die Sicherheit des neuronalen Schlüsselaustauschs verbessert werden.

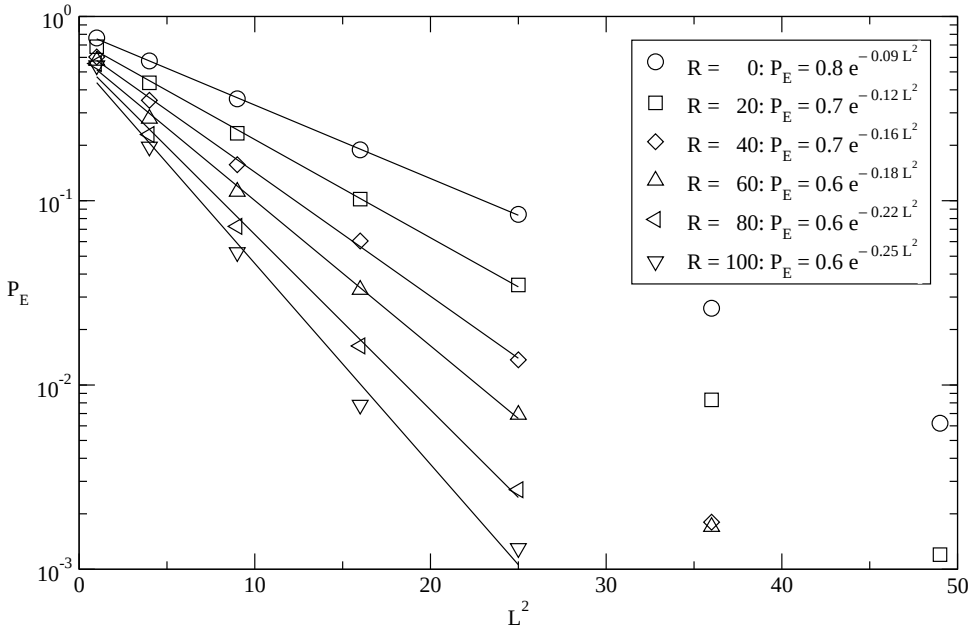


Abbildung 5.4: Erfolgswahrscheinlichkeit des geometrischen Angriffs für $K = 3$ und $N = 1000$ bei Verwendung der Anti-Hebb-Lernregel, ermittelt aus 10 000 Simulationen.

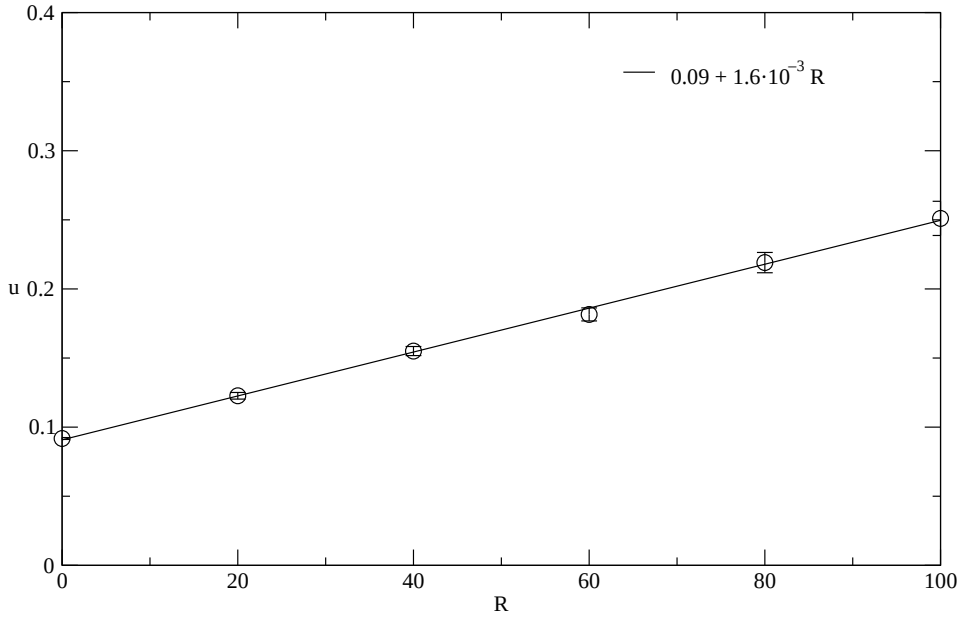


Abbildung 5.5: Einfluss des Feedback-Mechanismus auf den Erfolg des geometrischen Angriffs. Der Faktor u als Funktion von R wurde aus den in Abbildung 5.4 dargestellten Daten bestimmt. Die Fehlerbalken zeigen den bei der Regressionsanalyse ermittelten Standardfehler.

5.3 Iterative Rechnung für $N \to \infty$

Nachdem die Wirkung der Rückkopplung für kleine Systeme mittels Simulationen untersucht wurde, soll nun auch der Grenzfall $N \to \infty$ mit der iterativen Rechnung genauer betrachtet werden. Dazu muss aber der Feedback-Mechanismus geändert werden. Denn bisher wurde in jedem Zeitschritt nur ein neues Eingabeelement pro Zwischenschichtneuron erzeugt. In diesem Fall bleibt die Rückkopplung für $N \to \infty$ wirkungslos, weil nur endlich viele x_{ij} betroffen sind.

Deshalb wird für die iterative Rechnung angenommen, dass in jedem Zeitschritt ΛKN neue Eingabeelemente pro Tree Parity Machine erzeugt werden. In einer Simulation kann das beispielsweise so realisiert werden, dass jede der K Ausgaben σ_i mit ΛN Zufallswerten $z \in \{+1, -1\}$ multipliziert wird. Auf diese Weise sollte der Effekt des Feedback-Mechanismus auch für $N \to \infty$ beobachtbar sein. Im Folgenden wird immer $\Lambda = 10^{-3}$ verwendet, damit die Ergebnisse mit Simulationen zu $N = 1000$ verglichen werden können.

5.3.1 Rückkopplung bei der Synchronisation

Das in Kapitel 3.6.1 beschriebene Verfahren kann auch für $R > 0$ zur Bestimmung der Synchronisationszeit verwendet werden. Allerdings ist zu berücksichtigen, dass die von A und B verwendeten Eingaben nun voneinander abweichen können. In der iterativen Rechnung wird dieser Effekt durch den Eingabefehler λ_i beschrieben.

Die Größe λ_i ist definiert als der Anteil der Eingabeneuronen in der i-ten versteckten Einheit, für die $x_{ij}^A \neq x_{ij}^B$ gilt. Zu Beginn der Synchronisation sind alle $\lambda_i = 0$. Für die Änderung des Eingabefehlers im Verlauf des neuronalen Schlüsselaustauschs findet man:

$$\lambda_i^+ = (1 - \Lambda)\,\lambda_i + \Lambda\,\Theta(-\sigma_i^A \sigma_i^B)\,\Theta(\tau^A \tau^B)\,. \tag{5.7}$$

Außerdem erfolgt nach R Schritten mit $\tau^A \neq \tau^B$ ein Reset, bei dem die λ_i auf Null gesetzt werden.

Die Eingabefehler beeinflussen natürlich die Ausgaben σ_i der versteckten Einheiten. So hat ein Eingabeelement mit $x_{ij}^B = -x_{ij}^A$ dieselbe Wirkung wie ein Vorzeichenwechsel des Gewichts w_{ij}^B. Folglich kann λ_i bei der Berechnung des Verallgemeinerungsfehlers ϵ_i berücksichtigt werden, indem ρ_i durch einen effektiven Überlapp $\rho_{i,\text{eff}}$ ersetzt wird:

$$\rho_{i,\text{eff}} = (1 - 2\lambda_i)\rho_i\,. \tag{5.8}$$

Wenn nun $\rho_{i,\text{eff}}$ statt ρ_i verwendet wird, kann die Ausgabe der Tree Parity Machines wie in Kapitel 3.6.1 berechnet werden.

Auch die Formel, die die Anpassung der Gewichte in einem Synchronisationsschritt beschreibt, ändert sich für $R > 0$. Hier tritt ein zusätzlicher Term

proportional zu λ_i auf, der die repulsive Wirkung des Feedback-Mechanismus beschreibt:

$$\begin{aligned} f^{i+}_{a,b} &= \frac{1-\lambda_i}{2} \left(f^i_{a+\Delta^A_i, b+\Delta^B_i} + f^i_{a-\Delta^A_i, b-\Delta^B_i} \right) \\ &+ \frac{1}{2}\lambda_i \left(f^i_{a+\Delta^A_i, b-\Delta^B_i} + f^i_{a-\Delta^A_i, b+\Delta^B_i} \right) . \end{aligned} \tag{5.9}$$

Da in dieser Gleichung die Randbedingung $|w_{ij}| \leq L$ nicht berücksichtigt ist, kann sie nur für $-L < a, b < +L$ verwendet werden. Formeln für die Berechnung der Randelemente sind in Anhang B zu finden.

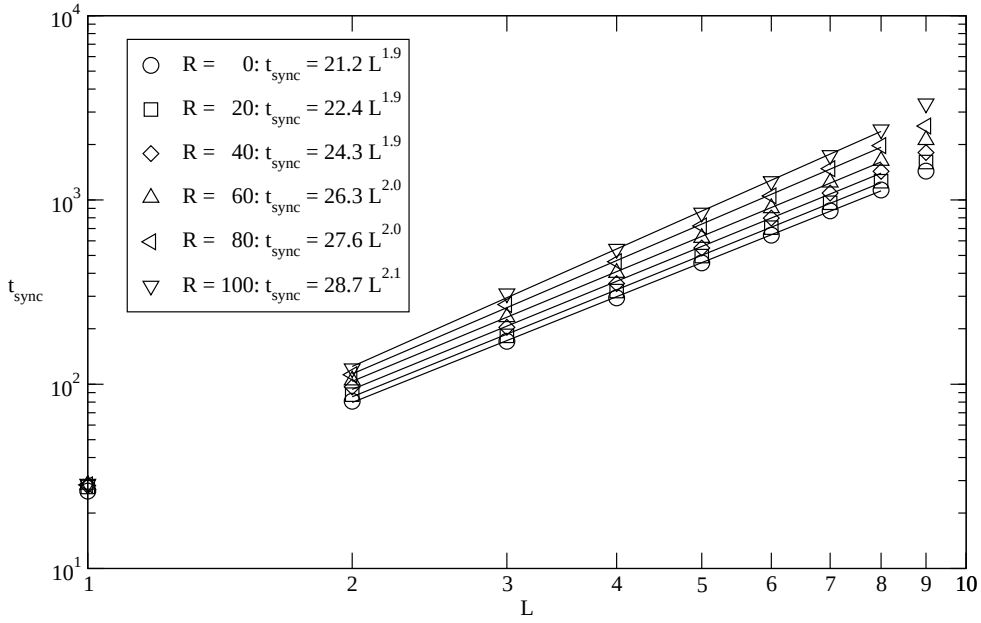


Abbildung 5.6: Zahl der benötigten Schritte bis zum Erreichen des Synchronisationskriteriums $\rho \geq 0.9$, gemittelt über 10 000 Berechnungen für $K = 3$.

Zur Bestimmung der Synchronisationszeit wird wie in Kapitel 3.6.1 die Bedingung $\rho \geq 0.9$ als Synchronisationskriterium verwendet. Das Ergebnis der iterativen Rechnung ist in Abbildung 5.6 zu sehen. Offensichtlich wächst $\langle t_{sync} \rangle$ auch für $R > 0$ proportional zu L^2 an:

$$\langle t_{sync} \rangle = cL^2 . \tag{5.10}$$

Allerdings beobachtet man im Vergleich zur Simulation (siehe Abbildung 5.2) keine Abweichung von diesem Verhalten für kleine L. Dies kann natürlich am Synchronisationskriterium liegen, wenn sich die Angleichung der Gewichte erst bei Erreichen eines größeren Überlapps als $\rho = 0.9$ verzögert. Andererseits vernachlässigt die iterative Rechnung, dass Fehler in der Eingabe nicht alle x_{ij} gleichmäßig betreffen. So wird in einer Simulation $x^A_{i1} \neq x^B_{i1}$ wesentlich häufiger

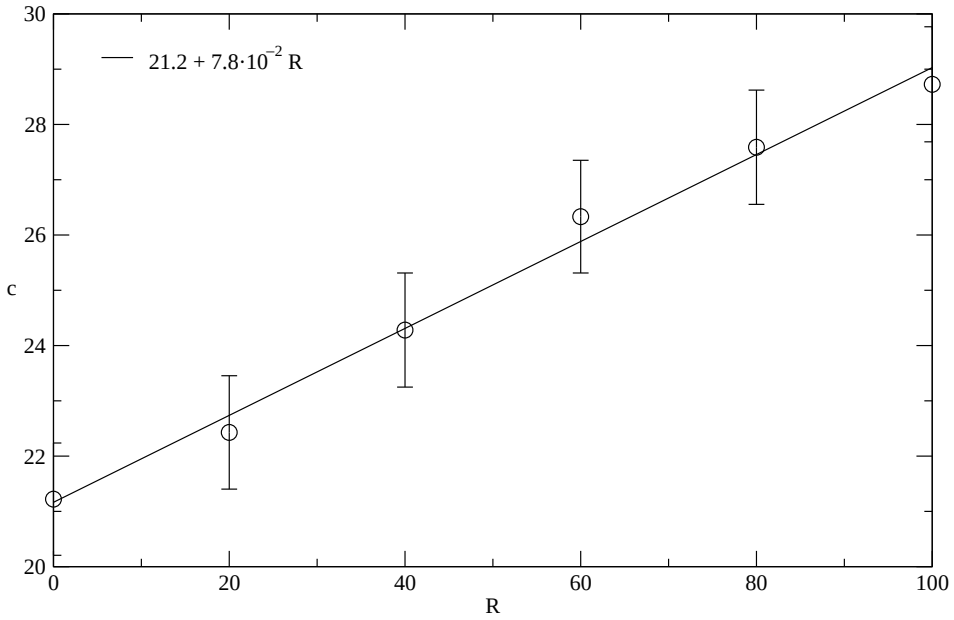


Abbildung 5.7: Verlängerung der Synchronisationszeit durch den Feedback-Mechanismus. Der Proportionalitätsfaktor c als Funktion von R wurde aus den in Abbildung 5.6 dargestellten Daten berechnet.

auftreten als $x_{iN}^A \neq x_{iN}^B$. Dies kann ebenfalls die Ursache für den beobachteten Unterschied zwischen iterativer Rechnung und Simulation sein.

Die repulsive Wirkung der Rückkopplung verzögert auch für $N \to \infty$ die Synchronisation zweier Tree Parity Machines. In Abbildung 5.7 ist gut zu erkennen, dass zwischen R und dem Proportionalitätsfaktor c aus Gleichung (5.10) ein linearer Zusammenhang besteht:

$$c(R) - c(0) \propto R\,. \tag{5.11}$$

Durch den Einsatz des Feedback-Mechanismus erhöht sich also der Aufwand für den neuronalen Schlüsselaustausch. Deshalb ist die Verwendung der Rückkopplung nur sinnvoll, wenn die Komplexität eines erfolgversprechenden Angriffs stärker als linear mit R anwächst. Dies soll nun für $N \to \infty$ überprüft werden.

5.3.2 ... und beim geometrischen Angriff

Die Wirkung des Feedback-Mechanismus auf die Erfolgswahrscheinlichkeit P_E des geometrischen Angriffs kann ebenfalls mit der iterativen Rechnung untersucht werden. Dazu ist es aber erforderlich, die Berechnung wie in Kapitel 3.6.2 mit drei beteiligten Tree Parity Machines durchzuführen.

Die für $R > 0$ möglichen Unterschiede zwischen den Eingaben der neuronalen Netze von A, B und E werden durch einen Satz von Eingabefehlern λ_i^m

berücksichtigt. Dabei gibt λ_i^m den Anteil der Eingabeelemente x_{ij}^m in der i-ten versteckten Einheit von m ($m \in \{A, B, E\}$) an, die von den korrespondierenden x_{ij} der anderen beiden neuronalen Netze abweichen. Aus dieser Definition folgt für die Änderung der Eingabefehler in einem Zeitschritt:

$$\lambda_i^{A+} = (1-\Lambda)\,\lambda_i^A + \Lambda\,\Theta(-\sigma_i^A\sigma_i^B)\,\Theta(-\sigma_i^A\sigma_i^E)\,\Theta(\tau^A\tau^B)\,; \tag{5.12}$$

$$\lambda_i^{B+} = (1-\Lambda)\,\lambda_i^B + \Lambda\,\Theta(-\sigma_i^B\sigma_i^A)\,\Theta(-\sigma_i^B\sigma_i^E)\,\Theta(\tau^A\tau^B)\,; \tag{5.13}$$

$$\lambda_i^{E+} = (1-\Lambda)\,\lambda_i^E + \Lambda\,\Theta(-\sigma_i^E\sigma_i^A)\,\Theta(-\sigma_i^E\sigma_i^B)\,\Theta(\tau^A\tau^B)\,. \tag{5.14}$$

Zu Beginn der Synchronisation werden alle Eingabefehler auf Null gesetzt. Ebenso erfolgt ein Reset nach R Schritten mit $\tau^A \neq \tau^B$.

Bei der Berechnung der lokalen Felder und der Zustände der versteckten Einheiten werden die Ordnungsparameter R_i^{AB}, R_i^{AE} und R_i^{BE} durch entsprechende effektive Größen ersetzt, in die auch die Eingabefehler eingehen:

$$R_{i,\text{eff}}^{mn} = (1 - 2\lambda_i^m - 2\lambda_i^n) R_i^{mn}\,. \tag{5.15}$$

Dabei kennzeichnen die Indizes m und n, zu welchem der Beteiligten die Tree Parity Machine gehört ($m, n \in \{A, B, E\}$).

Auch die Gleichung (3.48) für die Anpassung der Gewichte muss geändert werden, um die repulsive Wirkung der Rückkopplung zu berücksichtigen. Für die neuen Nicht-Randelemente f_{abe}^{i+} mit $-L < a, b, e < +L$ gilt dann:

$$\begin{aligned} f_{a,b,e}^{i+} &= \frac{1-\lambda_i^A-\lambda_i^B-\lambda_i^E}{2} \left(f^i_{a+\Delta_i^A,b+\Delta_i^B,e+\Delta_i^E} + f^i_{a-\Delta_i^A,b-\Delta_i^B,e-\Delta_i^E}\right) \\ &+ \frac{1}{2}\lambda_i^A \left(f^i_{a-\Delta_i^A,b+\Delta_i^B,e+\Delta_i^E} + f^i_{a+\Delta_i^A,b-\Delta_i^B,e-\Delta_i^E}\right) \\ &+ \frac{1}{2}\lambda_i^B \left(f^i_{a+\Delta_i^A,b-\Delta_i^B,e+\Delta_i^E} + f^i_{a-\Delta_i^A,b+\Delta_i^B,e-\Delta_i^E}\right) \\ &+ \frac{1}{2}\lambda_i^E \left(f^i_{a+\Delta_i^A,b+\Delta_i^B,e-\Delta_i^E} + f^i_{a-\Delta_i^A,b-\Delta_i^B,e+\Delta_i^E}\right)\,. \end{aligned} \tag{5.16}$$

Für die Randelemente ergeben sich ähnliche Formeln, in denen aber zusätzlich die Randbedingung $|w_{ij}| \leq L$ berücksichtigt wird.

Wie in Kapitel 3.6.2 gilt auch hier ein Angreifer als erfolgreich, wenn während der Iteration $\rho^{AE} \geq 0.9$ oder $\rho^{BE} \geq 0.9$ zu einem früheren Zeitpunkt erreicht wird als $\rho^{AB} \geq 0.9$. Abbildung 5.8 zeigt, dass die Erfolgswahrscheinlichkeit P_E eines geometrischen Angriffs exponentiell mit L abnimmt:

$$P_E \propto e^{-yL}\,. \tag{5.17}$$

Dies konnte auch schon in Kapitel 3.6.2 für $R = 0$ beobachtet werden. Hinzu kommt aber, dass der Vorfaktor y im Exponenten von der Stärke der Rückkopplung abhängt. In Abbildung 5.9 ist deutlich zu erkennen, dass der funktionale Zusammenhang zwischen y und R linear ist:

$$y(R) - y(0) \propto R\,. \tag{5.18}$$

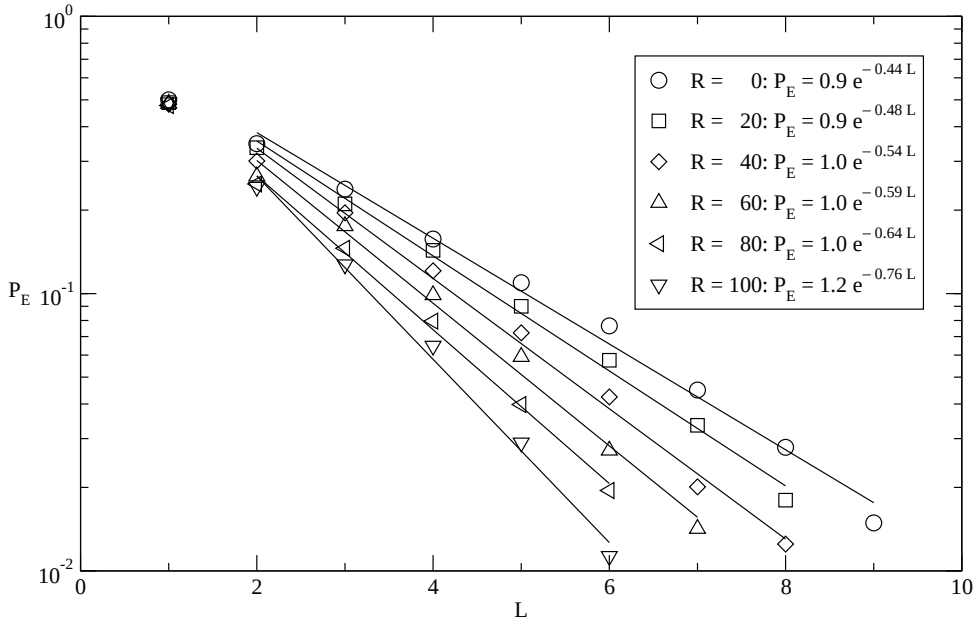


Abbildung 5.8: Erfolgswahrscheinlichkeit des geometrischen Angriffs, ermittelt aus 10 000 Berechnungen für $K = 3$.

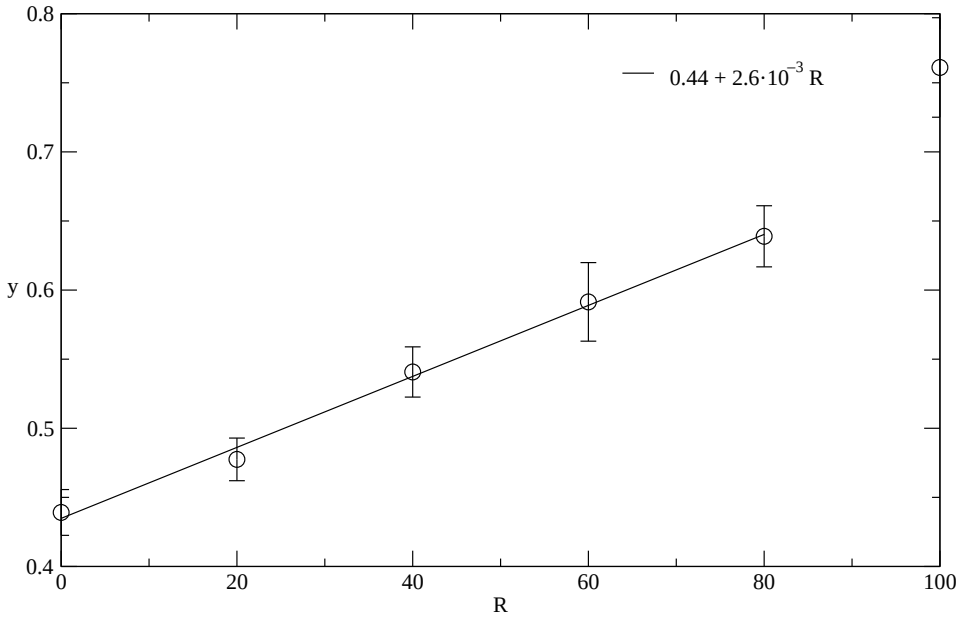


Abbildung 5.9: Verbesserung der Sicherheit durch den Einsatz des Feedback-Mechanismus. Der Faktor y als Funktion von R wurde aus den in Abbildung 5.8 dargestellten Daten berechnet.

5.3.3 Fazit

Die iterative Rechnung zeigt, dass der Feedback-Mechanismus das grundlegende Skalenverhalten von t_{sync} und P_E nicht verändert. Wie beim Schlüsselaustausch ohne Rückkopplung gilt $\langle t_{sync} \rangle = cL^2$ und $P_E \propto e^{-yL}$ auch für $R > 0$. Allerdings hängen die Faktoren c und y nun von der Stärke des Feedbacks ab. Deshalb führt eine Vergrößerung von R zum exponentiellen Abfall von P_E, während der Aufwand für die Kommunikationspartner nur linear anwächst.

Dieser Zusammenhang ist besonders deutlich zu erkennen, wenn die Erfolgswahrscheinlichkeit des geometrischen Angriffs als Funktion der mittleren Synchronisationszeit dargestellt wird (siehe Abbildung 5.10). Dabei stellt man fest, dass die Sicherheit des Schlüsselaustauschs bei gleicher Synchronisationszeit verbessert werden kann, indem L und R geeignet gewählt werden. Alternativ ist auch eine Reduktion des Aufwands bei gleichem Sicherheitsniveau möglich.

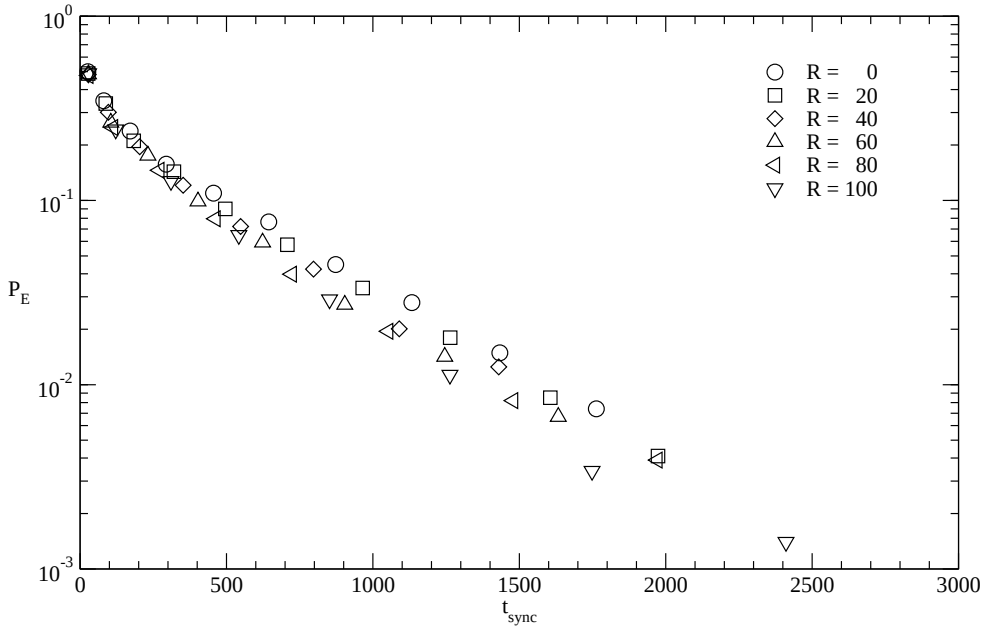


Abbildung 5.10: P_E als Funktion von $\langle t_{sync} \rangle$ für verschiedene Werte von R

Kapitel 6

Zusammenfassung und Ausblick

6.1 Ergebnisse

In dieser Arbeit wurde der kryptographische Schlüsselaustausch mit neuronalen Netzen untersucht und zum ersten Mal um einen Feedback-Mechanismus erweitert. Dabei ergab sich eine Reihe von zum Teil überraschenden Ergebnissen:

- Die Synchronisation zweier diskreter Tree Parity Machines kann als Random Walk der Gewichte beschrieben werden. Wegen der reflektierenden Randbedingungen haben koordinierte Lernschritte eine attraktive Wirkung, die den Überlapp erhöht.

- Da die Zustände der versteckten Einheiten nicht nach außen sichtbar sind, treten für $K > 1$ auch repulsive Schritte auf, die Synchronisation und Lernen verzögern. Der Vergleich mit $K = 1$ zeigt, dass diese Eigenschaft für die Sicherheit des neuronalen Schlüsselaustauschs notwendig ist.

- Die Wahl der Lernregel hat Auswirkungen auf die Länge der Gewichtsvektoren. Bei der Hebbschen Lernregel werden sie in der Anfangsphase der Synchronisation länger, bei der Anti-Hebb-Lernregel kürzer. Wenn die Random-Walk-Lernregel verwendet wird oder $L \ll \sqrt{N}$ gilt, bleibt die Länge der Gewichtsvektoren nahezu konstant.

- Zwei Tree Parity Machines, die gegenseitig ihre Ausgaben lernen, erreichen eine vollständige Synchronisation der Gewichte nach endlich vielen Schritten. Die mittlere Synchronisationszeit wächst proportional zu L^2 an.

- Während des neuronalen Schlüsselaustauschs bleiben die Gewichte der beteiligten Tree Parity Machines annähernd gleichverteilt.

- Das Lernen durch Zuhören läuft langsamer ab als die Synchronisation, weil die Häufigkeit repulsiver Schritte höher ist. Dieser Nachteil kann aber durch den Einsatz verbesserter Angriffsmethoden teilweise kompensiert werden.

- Die Erfolgswahrscheinlichkeit des geometrischen Angriffs fällt exponentiell mit zunehmendem L ab.
- Falls die Bedingung $L \ll \sqrt{N}$ nicht erfüllt ist und die Anti-Hebb-Lernregel eingesetzt wird, beobachtet man, dass die Erfolgswahrscheinlichkeit des geometrischen Angriffs exponentiell mit L^2 abfällt. Diese Änderung des Skalenverhaltens ist auf die im Vergleich zu anderen Lernregeln geringe Länge der Gewichtsvektoren zurückzuführen.
- Der höhere Aufwand des genetischen Angriffs lohnt sich nicht, weil ein Angreifer mit der geometrischen Angriffsmethode mehr Erfolg hat.
- Die Verwirrte Tree Parity Machine erzeugt eine pseudozufällige Bitsequenz, die für Verschlüsselungszwecke geeignet ist. Eine weitere Tree Parity Machine kann diese Zeitreihe nicht vorhersagen.
- Der Feedback-Mechanismus der Verwirrten Tree Parity Machine kann nur dann in das Schlüsselaustauschprotokoll integriert werden, wenn seine stark repulsive Wirkung auf geeignete Weise abgeschwächt wird.
- Durch den Einsatz der Rückkopplung lässt sich die Sicherheit des neuronalen Schlüsselaustauschs verbessern.
- Der Grenzfall $N \to \infty$ kann mit einer iterativen Berechnungsmethode untersucht werden. Die Resultate sind in guter Übereinstimmung mit den Ergebnissen der Simulationen.

6.2 Ausblick

Bei diesen Untersuchungen wurden nur einige Probleme aus dem Gebiet der *Neuronalen Kryptographie* betrachtet, das noch viele Fragestellungen bereithält.

Einerseits lassen sich weitere Ideen finden, mit denen eventuell eine Verbesserung der Sicherheit beim neuronalen Schlüsselaustausch möglich ist. Daraus kann man neue Varianten des Verfahrens entwickeln und deren Eigenschaften durch Simulationen bestimmen.

Andererseits ist es auch interessant, den Mechanismus der Synchronisation genauer zu untersuchen. Denn nur so lassen sich Erkenntnisse über die prinzipielle Sicherheit des Schlüsselaustauschs gewinnen. Beispielsweise ist die genaue Zahl der Schlüssel, die durch die Synchronisation zweier Tree Parity Machines erzeugt werden können, noch unbekannt. Auch wurde bisher nicht bewiesen, dass es keine Angriffsmethode gibt, die in polynomialer Zeit zum Erfolg führt.

Anhang A

Bezeichnungen

A	Sender
B	Empfänger
E	Angreifer
K	Anzahl der versteckten Einheiten pro Tree Parity Machine
L	Wertebereich der Gewichte ($\|w_{ij}\| \leq L$)
N	Anzahl der Gewichte je Zwischenschichtneuron
R	Parameter für den Feedback-Mechanismus
Λ	Parameter für die iterative Rechnung mit Rückkopplung
$\vec{w}_i$	Gewichtsvektor der i-ten versteckten Einheit
$\vec{x}_i$	Eingabevektor der i-ten versteckten Einheit
w_{ij}	j-te Komponente von $\vec{w}_i$
x_{ij}	j-te Komponente von $\vec{x}_i$
σ_i	Ausgabe des i-ten Zwischenschichtneurons
τ	Gesamtausgabe der Tree Parity Machine
h_i	lokales Feld in der i-ten versteckten Einheit
ρ_i	Überlapp in der i-ten versteckten Einheit
ρ	Mittelwert der ρ_i
ϵ	Verallgemeinerungsfehler
λ	Eingabefehler (siehe Kapitel 5.3.1)
t_{sync}	Synchronisationszeit
P_E	Erfolgswahrscheinlichkeit eines Angriffs
S	Entropie
$\mathcal{F}_i$	Matrix zur Beschreibung der Gewichtskonfiguration in der i-ten versteckten Einheit
f^i_{ab}	Element von $\mathcal{F}_i$

Anhang B

Ergänzungen zur iterativen Rechnung

B.1 Attraktive Schritte

Ein attraktiver Lernschritt findet statt, wenn $\tau^A = \tau^B = \sigma_i^A = \sigma_i^B$ gilt. In diesem Fall kann die Änderung der Matrixelemente f_{ab}^i, bei denen beide Indizes auf dem Rand liegen, durch folgende Formeln beschrieben werden:

$$f_{L,L}^{i+} = \frac{1}{2}(1-\lambda_i)\left(f_{L-1,L-1}^i + f_{L-1,L}^i + f_{L,L-1}^i + f_{L,L}^i\right) ; \quad \text{(B.1)}$$

$$f_{-L,-L}^{i+} = \frac{1}{2}(1-\lambda_i)\left(f_{-L+1,-L+1}^i + f_{-L+1,-L}^i + f_{-L,-L+1}^i + f_{-L,-L}^i\right) ; \quad \text{(B.2)}$$

$$f_{L,-L}^{i+} = \frac{1}{2}\lambda_i\left(f_{L-1,-L+1}^i + f_{L-1,-L}^i + f_{L,-L+1}^i + f_{L,-L}^i\right) ; \quad \text{(B.3)}$$

$$f_{-L,L}^{i+} = \frac{1}{2}\lambda_i\left(f_{-L+1,L-1}^i + f_{-L+1,L}^i + f_{-L,L-1}^i + f_{-L,L}^i\right) . \quad \text{(B.4)}$$

Bei der Synchronisation ohne Feedback ist der Eingabefehler λ_i in diesen Gleichungen auf Null zu setzen. Für Matrixelemente f_{ab}^i, bei denen nur ein Index auf dem Rand liegt ($a = \pm L$ und $b \neq \pm L$ oder umgekehrt), gilt:

$$f_{a,L}^{i+} = \frac{1-\lambda_i}{2}\left(f_{a-1,L}^i + f_{a-1,L-1}^i\right) + \frac{1}{2}\lambda_i\left(f_{a+1,L}^i + f_{a+1,L-1}^i\right) ; \quad \text{(B.5)}$$

$$f_{a,-L}^{i+} = \frac{1-\lambda_i}{2}\left(f_{a+1,-L}^i + f_{a+1,-L+1}^i\right) + \frac{1}{2}\lambda_i\left(f_{a-1,-L}^i + f_{a-1,-L+1}^i\right) ; \quad \text{(B.6)}$$

$$f_{L,b}^{i+} = \frac{1-\lambda_i}{2}\left(f_{L,b-1}^i + f_{L-1,b-1}^i\right) + \frac{1}{2}\lambda_i\left(f_{L,b+1}^i + f_{L-1,b+1}^i\right) ; \quad \text{(B.7)}$$

$$f_{-L,b}^{i+} = \frac{1-\lambda_i}{2}\left(f_{-L,b+1}^i + f_{-L+1,b+1}^i\right) + \frac{1}{2}\lambda_i\left(f_{-L,b-1}^i + f_{-L+1,b-1}^i\right) . \quad \text{(B.8)}$$

Die Änderung der Matrixelemente f_{ab}^i mit $-L < a, b < +L$ wurde bereits in den Kapiteln 3.6.1 und 5.3.1 angegeben.

B.2 Repulsive Schritte

Ein repulsiver Lernschritt findet statt, wenn $\tau^A = \tau^B$, aber $\sigma_i^A \neq \sigma_i^B$ gilt. In diesem Fall spielt der Eingabefehler λ_i auch bei der Synchronisation mit Feedback keine Rolle.
Für $\tau^B \neq \sigma_i^B$ ändern sich nur die Gewichte im neuronalen Netz von A:

$$f_{a,b}^{i+} = \frac{1}{2}(f_{a+1,b}^i + f_{a-1,b}^i)\,; \tag{B.9}$$

$$f_{L,b}^{i+} = \frac{1}{2}(f_{L,b}^i + f_{L-1,b}^i)\,; \tag{B.10}$$

$$f_{-L,b}^{i+} = \frac{1}{2}(f_{-L+1,b}^i + f_{-L,b}^i)\,. \tag{B.11}$$

Gleichung (B.9) gilt für $a \neq \pm L$.
Wenn aber $\tau^A \neq \sigma_i^A$ gilt, werden nur die Gewichte in der Tree Parity Machine von B angepasst:

$$f_{a,b}^{i+} = \frac{1}{2}(f_{a,b+1}^i + f_{a,b-1}^i)\,; \tag{B.12}$$

$$f_{a,L}^{i+} = \frac{1}{2}(f_{a,L}^i + f_{a,L-1}^i)\,; \tag{B.13}$$

$$f_{a,-L}^{i+} = \frac{1}{2}(f_{a,-L+1}^i + f_{a,-L}^i)\,. \tag{B.14}$$

Gleichung (B.12) gilt für $b \neq \pm L$.

Literaturverzeichnis

[1] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[2] R. Metzler, W. Kinzel, and I. Kanter. Interacting neural networks. *Phys. Rev. E*, 62(2):2555–2565, 2000.

[3] I. Kanter, W. Kinzel, and E. Kanter. Secure exchange of information by synchronization of neural networks. *Europhys. Lett.*, 57(1):141–147, 2002.

[4] A. Beutelspacher. *Kryptologie*. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 2002.

[5] R. Metzler, W. Kinzel, L. Ein-Dor, and I. Kanter. Generation of unpredictable time series by a neural network. *Phys. Rev. E*, 63:056126, 2001.

[6] W. Kinzel. Theory of Interacting Neural Networks. cond-mat/0204054, 2002.

[7] W. Kinzel and I. Kanter. Neural Cryptography. cond-mat/0208453, 2002.

[8] A. Klimov, A. Mityaguine, and A. Shamir. Analysis of Neural Cryptography. In *ASIACRYPT*, 2002.

[9] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel. Secure key-exchange protocol with an absence of injective functions. *Phys. Rev. E*, 66:066102, 2002.

[10] M. Rosen-Zvi, I. Kanter, and W. Kinzel. Cryptography based on neural networks—analytical results. *J. Phys. A*, 35:L707–L713, 2002.

[11] M. Rosen-Zvi, E. Klein, I. Kanter, and W. Kinzel. Mutual learning in a tree parity machine and its application to cryptography. *Phys. Rev. E*, 66:066135, 2002.

[12] W. Kinzel and I. Kanter. Interacting neural networks and cryptography. cond-mat/0203011, 2002.

[13] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, 1999.

[14] M. Rosen-Zvi and I. Kanter. Training a perceptron in a discrete weight space. *Phys. Rev. E*, 64:046109, 2001.

[15] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[16] G. Reents and R. Urbanczik. Self-Averaging and On-Line Learning. *Phys. Rev. Lett.*, 80(24):5445–5448, 1998.

[17] A. K. Hartmann and H. Rieger. A practical guide to computer simulations. cond-mat/0111531, 2001.

[18] E. Eisenstein, I. Kanter, D. A. Kessler, and W. Kinzel. Generation and Prediction of Time Series by a Neural Network. *Phys. Rev. Lett.*, 74(1):6–9, 1995.

[19] H. Zhu and W. Kinzel. Antipredictable Sequences: Harder to Predict Than Random Sequences. *Neural Computation*, 10(8):2219–2230, 1998.

[20] R. Metzler. Online-Lernen wechselwirkender neuronaler Netze. Diplomarbeit, Universität Würzburg, 1999.

[21] W. Bialek, I. Nemenman, and N. Tishby. Predictability, Complexity and Learning. *Neural Computation*, 13(11):2409–2463, 2001.

[22] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1981.

Danksagung

Viele Menschen haben auf unterschiedlichste Art und Weise zum Gelingen dieser Arbeit beigetragen. Ihnen gebührt hiermit mein herzlichster Dank:

- Prof. Dr. Wolfgang Kinzel für die interessante Aufgabenstellung und die sehr gute Betreuung. Seine Vorschläge haben die Richtung dieser Arbeit entscheidend geprägt.
- Priv. Doz. Dr. Michael Biehl und Prof. Dr. Georg Reents für kompetenten Rat zu den immer wieder auftretenden Fragen und Problemen.
- Prof. Ido Kanter für die interessanten Diskussionen während seiner Aufenthalte hier in Würzburg. Seine Anregungen waren sehr wertvoll.
- Florian Much und Thorsten Volkmann für die Hilfe bei Computerproblemen und das sorgfältige Korrekturlesen dieser Arbeit.
- den Systembetreuern Prof. Dr. Georg Reents, Christopher Dahnken, Andreas Vetter und Alexander Wagner für die hervorragende Wartung des Computersystems.
- den Sekretärinnen Christine Schmeisser, Brigitte Wehner und Uschi Eitelwein für die Hilfe bei bürokratischen Problemen.
- allen Mitgliedern des Lehrstuhls für die angenehme Arbeitsatmosphäre, die mir ein konstruktives und relativ entspanntes Arbeiten ermöglicht hat.
- Besonderen Dank schulde ich natürlich meiner Familie für die moralische und nicht zuletzt finanzielle Unterstützung während meines Studiums.

Erklärung

Diese Diplomarbeit wurde am Lehrstuhl für Theoretische Physik III (Computational Physics) der Universität Würzburg angefertigt. Die Computer-Programme wurden in den Programmiersprachen C und C++ realisiert. Die Simulationen wurden größtenteils auf den Workstations des Instituts durchgeführt. Für einige Berechnungen wurde auch das Programm „Mathematica“ verwendet. Die Abbildungen wurden mit „xmGrace“ und „xfig“ erstellt und die Arbeit mit Hilfe von LaTeX gesetzt. Ich habe mich bemüht, die Regeln der amtlichen neuen deutschen Rechtschreibung zur Anwendung zu bringen.

Gemäß 3 Absatz 2 der Diplomprüfungsordnung für den Diplom-Studiengang Physik an der Universität Würzburg erkläre ich hiermit, dass ich die Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Würzburg, den 8. Juli 2003

Andreas Ruttor