Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Klassifikation und Prädiktion von Trajektorien mit neuro-evolvierten RNNs

Ausarbeitung zum Hauptseminar

Manuel Zellhöfer

Betreuer: Dipl. Inf. Christian Vollmer
Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Inhaltsverzeichnis

1 Einführung **1**

2 Grundlagen **1**

2.1 Backpropagation through time (BPTT) 2

2.2 Real-time recurrent learning (RTRL) 3

2.3 Teacher Forcing 4

2.4 Neuroevolution 4

3 Aktuelle Entwicklungen **6**

3.1 Überblick 6

3.2 Echo State Networks (ESN) 7

3.3 Long Short-Term Memory (LSTM) 8

4 Verschwindender Fehlergradient und das „konstante Fehlerkarussell" **11**

4.1 Beschreibung des Problems 11

4.2 Der Lösungsansatz des LSTM 12

5 Neuroevolution für LSTM - EVOLINO **12**

5.1 Vebesserung des LSTM 13

5.2 Neuroevolution 14

5.3 Im Vergleich mit dem ESN 16

Fazit **16**

Literatur **17**

1 Einführung

An mobile Assistenzroboter, die mit Menschen in einer natürlichen Umgebung zusammenarbeiten sollen sind zahlreiche Anforderungen gestellt. Unter vielen anderen Punkten ist es beispielsweise wichtig, dass ein solcher Roboter in der Lage ist die Bewegung von Menschen und Objekten in seinem Umfeld vorauszusagen oder auch seine eigenen Bewegungen effizient vorherzusehen. Ähnlich wie der Mensch, der imstande ist aus Bewegungsmustern einer Person die weitere Bewegung vorherzusagen, könnte ein Roboter mit denselben 'Fähigkeiten' reagieren, interagieren und sogar von sich aus auf potentielle Nutzer zugehen. [Hellbach et al., 2008] [Wierstra et al., 2005]

In dieser Arbeit soll zuerst ein überblick über existierende Methoden zur Zeitreihenvorhersage mit rekurrenten Neuronalen Netzen (RNN, auch „dynamische neuronale Netze") geschaffen werden. Die Zeitreihenanalyse und -vorhersage ist ein weit erforschtes Feld im Bereich der Signalverarbeitung. RNNs sind ein wichtiges Werkzeug auf diesem Gebiet.

Desweiteren soll eine besondere Schwierigkeit des Trainings von RNNs, das Problem des verschwindenden Gradienten, erläutert werden. Das Long Short-Term Memory (LSTM), welches dieses Problem direkt anspricht, soll untersucht werden und mit den zurzeit wohl performantesten RNNs, den Echo State Networks (ESN) verglichen werden. Hierzu soll auch auf das Training des LSTM mittels EVOLINO (EVolution of recurrent systems with Optimal LINear Outputs), sozusagen ein evolutionäres Gerüst zur Gewichtseinstellung von RNNs, eingegangen werden.

2 Grundlagen

Die angesprochenen Bewegungsmuster oder -trajektorien werden in der Regel durch eine Reihe von Punkten im dreidimensionalen Raum dargestellt. Durch die Aufteilung in Komponenten lässt sich eine solche Trajektorie in verschiedenen eindimensionalen Zeitreihen darstellen, mit allen Vor- und Nachteilen die diese Vereinfachung mit sich bringt.

RNNs eignen sich aufgrund von Rückkopplungen in der Netzstruktur und damit ermöglichtem dynamischem Verhalten für die Vorhersage dynamischer Systeme. Nachgewiesenermaßen sind RNNs theoretisch sogar in der Lage jedes beliebige nichtlineare dynamische System zu modellieren. Diese Eigenschaft ist ausschlaggebend um RNNs zur Prädiktion von Bewegungstrajektorien in Betracht zu ziehen.[Mandic und Chambers, 2000],[Wierstra et al., 2005] Die große Herausforderung besteht darin, ein solches Netz dazu zu bringen, die gewünschte Dynamik zu zeigen - sprich ein RNN

erfolgreich zu trainieren. Beim Training per Gradientenabstieg tritt dabei das eingangs erwähnte und in Abschnitt 4.1 erläuterte Problem des verschwindenden Gradienten auf.

Bei den hier vorgestellten Lernalgorithmen handelt es sich um die ersten, gradientenbasierten Verfahren, die etwa Anfang der 1990er für RNNs vorgestellt wurden. Beide basieren auf dem selben Gradienten. Der Fehler der dabei zur Berechnung hergezogen wird, ist der Gesamtfehler über alle Zeitschritte:

$$F = \sum_{t=0}^{t_{max}} F(t)$$

mit $F(t)$:

$$F(t) = \frac{1}{2} \sum_{\text{k ist Ausgabeeinheit}} (t_k(t) - y_k(t))^2$$

und

$$\Delta w_{ij} = -\eta \frac{\partial F}{\partial w_{ij}}$$

lässt sich die Gewichtsänderung allgemein darstellen:

$$\Delta w_{ij} = -\eta \sum_{t=0}^{t_{max}} \frac{\partial F(t)}{\partial w_{ij}} = -\eta \sum_{t=0}^{t_{max}} \sum_{\tau=1}^{t} \frac{\partial F(t)}{\partial w_{ij}(t-\tau)}$$

Zu beachten ist, dass die Ableitung nach dem Gewicht zu jedem Zeitschritt unterschiedlich ist, da sich der interne Zustand des Netzes verändert.

Auf *Backpropagation through time* oder *Real-time tecurrent learning* baut der Großteil der in Abschnitt 3 vorgestellten Methoden auf. Auch werden Neuentwicklungen immer wieder mit diesen Verfahren verglichen.

In den übrigen beiden Abschnitten wird kurz ein allgemeines Verfahren zur Trainingsbeschleunigung vorgestellt sowie die Grundideen von Neuroevolution erläutert.

2.1 Backpropagation through time (BPTT)

Backpropagation through time ist wohl der erste vorgeschlagene Trainingsalgorithmus für RNNs. Er basiert auf dem Backpropagation Algorithmus für vorwärtsgerichtete Netze. Die Grundidee ist dabei, dass das RNN über die Zeit „aufgefaltet" wird. Das heißt, für jeden Zeitschritt werden Kopien des Netzes angelegt und -bildlich gesprochen- übereinandergelegt. Die Gewichte zwischen

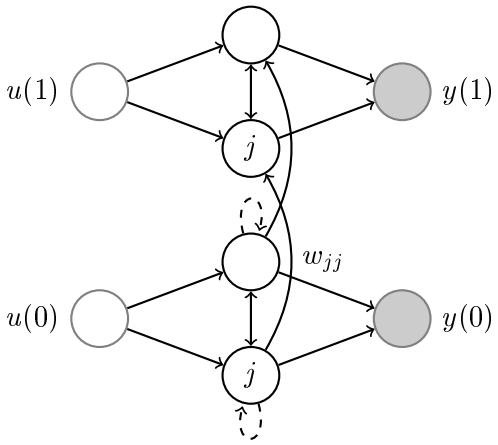

Abbildung 1: Zeitliche „Entfaltung" bei BPTT für ein einfaches RNN mit einem Ein- und Ausgang. Die gestrichelten Kanten deuten die rekurrenten Verbindungen an, die durch Verbindungen „in den nächsten Zeitschritt" ersetzt werden.

den Neuronen bleiben die gleichen, wodurch ein vorwärtsgerichtetes Netz entsteht, welches mit dem bekannten BP-Algorithmus trainiert wird. Abbildung 1 veranschaulicht diese Idee.

Bei der Berechnung des Gradienten mittels BPTT müssen also die vergangenen Aktivierungen aller Neuronen gespeichert werden um die Rückpropagierungsphase zu ermöglichen. Dies führt zu einem sehr hohen Speicherplatzbedarf.

Eine Abwandlung des BPTT ist das *truncated* BPTT. Dabei wird die Auffaltung nach n Zeitschritten abgebrochen. Die Gewichtsänderung kann dann wie folgt ausgedrückt werden:

$$\Delta w_{ij} = -\eta \sum_{t=t_{max}-n}^{t_{max}} \sum_{\tau=t_{max}-n}^{t} \frac{\partial F(t)}{\partial w_{ij}(t-\tau)}$$

[Zell, 1994], [Hochreiter, 1998]

2.2 Real-time recurrent learning (RTRL)

Eine andere herangehensweise ist das Real-time recurrent learning. Die Rückpropagierungsphase entfällt dabei, da hier Information die zur Gradientenberechnung nötig ist stetig in der Ausbrei-

tungsphase „gesammelt" werden kann :

$$\begin{aligned}
\Delta w_{ij} &= -\eta \frac{\partial F}{\partial w_{ij}} = -\eta \sum_{t=0}^{t_{max}} \frac{\partial F(t)}{\partial w_{ij}} = \sum_{t=0}^{t_{max}} \Delta w_{ij}(t) \\
\Delta w_{ij}(t) &= -\eta \frac{\partial F(t)}{\partial w_{ij}} \\
&= -\eta \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{\text{k ist Ausgabeeinheit}} (t_k(t) - y_k(t))^2 \\
&= -\eta \sum_{\text{k ist Ausgabeeinheit}} (t_k(t) - y_k(t)) \cdot \frac{\partial y_k(t)}{\partial w_{ij}}
\end{aligned}$$

Für jedes Gewicht und jede Ausgabeeinheit wird nun der Wert der Ableitung $\frac{\partial y_k(t)}{\partial w_{ij}}$ in einer Variablen gespeichert und für jeden Zeitschritt nur noch aktualisiert und zur Berechnung der Gewichtsänderung herangezogen. Die Variablen selbst werden zu Beginn gleich Null gesetzt. Mit dieser Methode ist es möglich das Netz „in Echtzeit" zu trainieren. Ein Vorteil gegenüber BPTT ist der über der Zeit konstante Speicherplatzverbrauch der sich theoretisch aber erst bei sehr langen Zeitabständen durchschlagen würde. Dieser Vorteil wird zusätzlich erkauft mit hoher Berechnungskomplexität. RTRL eignet sich folglich eher für kleine RNNs. [Schmidhuber, 1993], [Zell, 1994], [Jaeger, 2002]

2.3 Teacher Forcing

Teacher Forcing ist eine Technik, mit deren Hilfe das Training von RNNs beschleunigt werden kann. Man ersetzt dabei die Ausgaben des Netzes $y_k(t)$ mit den für diesen Zeitschritt gewünschte Ausgabe des Trainingsdatensatzes $t_k(t)$. Damit wird direkt in den Zustand des RNNs eingegriffen und das Netz auf die gewünschte Ausgabe „gezwungen". Man könnte dies je nachdem auch so Interpretieren, dass das RNN in einen Arbeitspunkt *verschoben* wird. Prinzipiell lässt sich Teacher Forcing auf jedes RNN anwenden. Wichtig ist nur, dass die Ausgabeneuronen auch rekurennte Verbindungen aufweisen, also mit dem übrigen Netz verbunden sind. [Zell, 1994], [Haykin, 2009]

2.4 Neuroevolution

Unter Neuroevolution versteht man die Anwendung von genetischen Algorithmen auf die Parameter- und Architekturbestimmung neuronaler Netze. Bei genetischen Algorithmen handelt es sich um

stochastische Such- oder Optimierungsmethoden, die grob nach den Prinzipien biologischer Evolution vorgehen („Survival of the Fittest"). Dabei werden einzelne Individuen einer Population von Lösungen mithilfe einer Fitness-Funktion evaluiert. Um diese Fitness nun zu maximieren wird die existierende Population durch verschiedene Operatoren verbessert. Die elementarsten sind

Selektion Aus der Population werden die besten Individuen ausgewählt. Dies geschieht in der Regel durch einen Auswahlprozess, der die jeweilige Fitness berücksichtigt.

Kreuzung Die Eigenschaften (an die Biologie angelehnt „Genotypen" genannt) zweier Individuen werden zu neuen Individuen rekombiniert.

Mutation Vorhandene Individuen werden zufälligen, geringfügigen Änderungen unterzogen.

[Pohlheim, 1999]

Für neuronale Netze gibt es nun unterschiedliche Ansätze evolutionäre Algorithmen zu verwenden. Diese reichen von der

- Einstellung der Gewichte (was einer globalen Trainingsmethode entspricht) über die
- Evolution der Netzarchitektur (die es einem Netz ermöglicht seine Topologie dem Problem anzupassen) bis hin zur
- Evolution der Lernregeln (was einer Art Meta-Lernen, „lernen-zu-lernen" gleichkommt)[1].

Zur jeweiligen Anwendung muss dann eine geeignete Repräsentation des neuronalen Netzes gewählt werden. Für den ersten Fall beispielsweise werden in der Regel alle Gewichte eines Individuums (hier: Netzes) durch einen reellen Vektor dargestellt, auf den dann die genannten Operatoren angewandt werden. Die Rekombination wäre im einfachsten Fall etwa das Austauschen von Elementen, Mutation könnte man durch addition eines Zufallsvektors erreichen. Die Wahl der Repräsentation und der Operatoren hängt hauptsächlich ab von der jeweiligen Problemstellung und dem Teilaspekt des Netzes, welcher evolviert werden soll.

Vorteile haben evolutionäre Algorithmen zur Gewichtseinstellung besonders dann, wenn Fehlerinformation zur Gradientenberechnung schwer zugänglich oder gar nicht verfügbar ist. Auch wenn die Fehlerfunktion viele lokale Minima aufweist, erweisen sich evolutionäre Algorithmen als nützlich. Die Anwendung auf die Architektur ermöglicht es kompakte Netze zu erzeugen und die Adaptivität zu erhöhen. [Yao, 1999]

[1] In [Yao, 1993] wurde etwa eine allgemeine Lernregel als Funktion von lokalen Variablen (wie Aktivierung und aktuelles Gewicht) definiert. Konstanten innerhalb dieser Funktion wurden dann evolviert.

Es gibt unzählige Möglichkeiten evolutionäre Algorithmen allgemein und in Kombination mit neuronalen Netzen anzuwenden. Hier soll darauf nicht weiter eingegangen werden. Einen ausführlichen Überblick bieten etwa [Floreano et al., 2008] oder [Yao, 1999]. Im nächsten Abschnitt 3.1 sind auch Beispiele zu neuroevolvierten Techniken bezüglich Zeitreihenanalyse genannt.

3 Aktuelle Entwicklungen

Im folgenden werden aktuelle Lernalgorithmen und Architekturen für RNNs gezeigt. Der erste Abschnitt soll dabei einen möglichst breiten Überblick über das Gebiet zeigen und dabei auch kurz auf die Anwendung und Güte der vorgestellten Methoden eingehen. Die Auflistung macht dabei keinen Unterschied, ob es sich um Algorithmen oder Architekturen handelt.

Die übrigen beiden Abschnitte befassen sich mit zwei vielversprechenden Technologien, den ESN bzw. dem LSTM. Zusätzlich zu Anwendungen und Güte wird hierbei auch genauer auf den jeweiligen Aufbau der Netze eingegangen.

3.1 Überblick

Stochastische Zustandsschätzung Eine bisweilen für RNNs angewandte Trainingsmethode basiert auf der Zustandsschätzung für nichtlineare Systeme durch Kalman-Filter (bzw. Extended Kalman-Filter). Hierbei werden Parameter (Gewichte) und evtl. Zustände (Aktivierung) geschätzt und entsprechend eingestellt. Diese Trainingsmethoden wurden erfolgreich in Bereichen wie Signalverarbeitung, Modellierung nichtlinearer Systeme (z.B. Bioreaktoren) oder Regelungsaufgaben (z.B. Motorsteuerung) eingesetzt. Verglichen mit BPTT und RTRL scheinen sie dabei schneller zu lernen und bessere Ergebnisse zu liefern. [Puskorius und Feldkamp, 1994], [Jaeger, 2002]

Ähnliche Ansätze nutzen den sog. EM-Algorithmus (Expectation-Maximization) um die Parameter eines RNNs einzustellen. Dabei handelt es sich um eine Art erweiterte Maximum-Likelihood Methode, die mit unvollständigen Datensätzen eine iterative Parameterschätzung durchführt. Die Methode wurde neben der Prädiktion eines synthetischen dynamischen Systems auch zur Prädiktion von Atembewegungen und der Vorhersage von Weltraumwetter erfolgreich eingesetzt. [Unkelbach et al., 2009], [Mirikitani und Ouarbya, 2009]

Hierarchische Netze Eine andere herangehensweise besteht darin, neuronale Netze hierarchisch zu Organisieren. Die Möglichkeiten neuronale Netze zu kombinieren sind sicher vielfältig. Hier seien zwei Beispiele genannt. In [Sumpter und Bulpitt, 2000] wird ein RNN hinter ein vorwärtsgerichtetes „Symbol Network" geschaltet. Damit wurden gute Ergebnisse bei der Prädiktion von Bewegungen in 2D Bildern erreicht.

[Namikawa und Tani, 2008] stellt eine Methode vor, bei der mehrere RNNs parallel trainiert werden. Jedes dieser RNNs spezialisiert sich dabei auf ein Teilproblem. Aus diesen RNNs wird dann mittels eines wiederum rekurrenten „Gating Networks" das am ehesten in Frage kommende „Experten"-RNN gewählt. Der Trainingsalgorithmus basiert dabei auf BPTT. Dieses Netz wurde auf zweidimensionale Lissajous-Kurven angewandt, bei welchen die Frequenz einer Markov-Kette folgend alternierte.

Neuroevolvierte Netze [Webb, 2008] nutzt Neuroevolution um Kaskaden von Echo State Netzwerken (siehe nächster Abschnitt) zu trainieren. Dabei werden nicht die Gewichte innerhalb eines ESN eingestellt sondern die Verbindungen zwischen einzelnen ESNs. Eine solche Kombination von ESNs ist in der Lage, die Vorhersage von Zeitreihen sukzessive zu Verbessern.

In [Qian-Li et al., 2008] wird eine Methode vorgestellt, die mittels Evolution gleichzeitig Vorhersageparameter (Einbettungsverzögerung, Einbettungsdimension) und Parameter von verwendeten RNNs zur Vorhersage (Anzahl an Hidden-Neuronen) einstellt. Die Netze selbst werden dann mit einem numerischen Optimierungsverfahren trainiert. Die Methode wurde verwendet um, neben synthetischen Zeitreihen, den Verlauf der Anzahl von Sonnenflecken vorherzusagen.

3.2 Echo State Networks (ESN)

Echo State Netzwerke sind RNNs bei welchen die Dynamiken, die durch die Rückkopplungen entstehen, auf ungewöhnliche Art und Weise ausgenutzt werden. Es besteht im Grunde aus einem rekurrenten Teil und einer linearen Ausgabeschicht. Der rekurrente Teil wird zu Beginn zufällig generiert. Da dieser im Idealfall eine reiche Vielfalt von Dynamiken aufweist wird er auch „dynamisches Reservoir" genannt.

Im Anschluss wird das Netz mittels Teacher Forcing (siehe Abschnitt 2.3) in die Nähe der gewünschten Dynamik „eingeschwungen". Die lineare Ausgabeschicht wird, sobald genügend Netzausgaben gesammelt wurden mittels Regression auf das jeweilige Problem eingestellt, die übrigen Gewichte

bleiben (in der ursprünglichen) Form konstant. Intuitiv erklärt „liest" die lineare Ausgabeschicht Dynamiken aus dem Reservoir aus und modelliert so die gewünschte Dynamik.

Die Beliebigkeit der rekurrenten Verbindungen im Reservoir ist natürlich nicht unbeschränkt. Um effektiv genutzt werden zu können muss dieses die sogenannte *echo state Eigenschaft* aufweisen. Dadurch entstehen gewisse Anforderungen an die Gewichtsmatrix auf die hier jedoch nicht näher eingegangen werden soll (siehe [Jaeger, 2002] oder [Lukosevicius und Jaeger, 2009]).

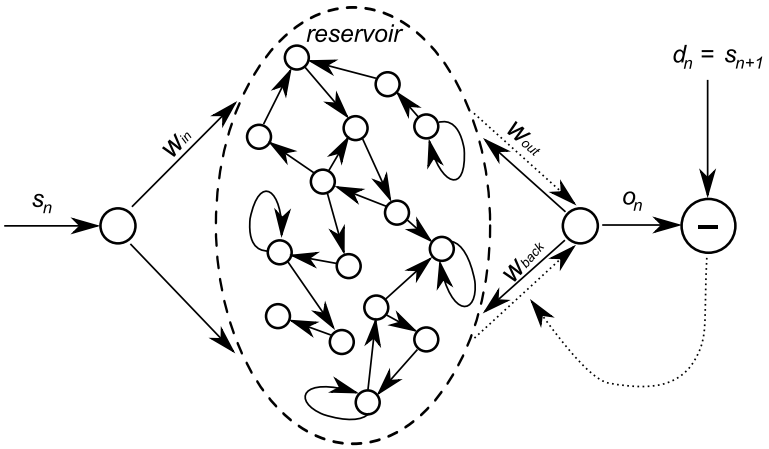

Abbildung 2: Das Echo State Netzwerk zur Zeitreihenvorhersage. Fixe Gewichte (etwa im Reservoir) sind mit durchzogenen Linien angedeutet. Die gestrichenen Gewichte werden mit linearer Regression eingestellt.

Ein funktionierendes ESN ist in der Lage mit recht hoher Verarbeitungsgeschwindigkeit sehr genaue Vorhersagen zu treffen. ESN fanden in der Zeitreihenprädiktion bereits breite Anwendung: Etwa bei der Modellierung von Strömungsverhalten ([Ishu et al., 2004]), eines nichtlinearen Mobilfunkkanals ([Jaeger, 2003]). Auch zur Vorhersage von Bewegunstrajektorien wurden ESNs bereits verwendet ([Hellbach et al., 2008]), was der Hauptgrund für ihre Betrachtung in dieser Arbeit ist.

Eine ausführliche Übersicht über das u.a. mit dem ESN aufgekommene Prinzip des „Reservoir Computing" bietet [Lukosevicius und Jaeger, 2009]. Reservoir Computing erfreut sich in letzter Zeit rasch wachsender Beliebtheit. [Reservoir Computing, 2010] [Projekt PHOCUS, 2010]

3.3 Long Short-Term Memory (LSTM)

Der Hauptbestandteil eines LSTMs ist die sogenannte Memory-Cell. Der Aufbau einer solchen Memory-Cell ist in Abbildung 3 dargestellt. Sie besitzt einen Zustand (S) welcher direkt zurückge-

koppelt ist. Dieses Konstrukt wird „konstantes Fehlerkarussell" genannt (näheres in Abschnitt 4.2). Ein weiterer Bestandteil sind die sogenannten Gates (Input- und Output-Gate, G_i, bzw. G_o).

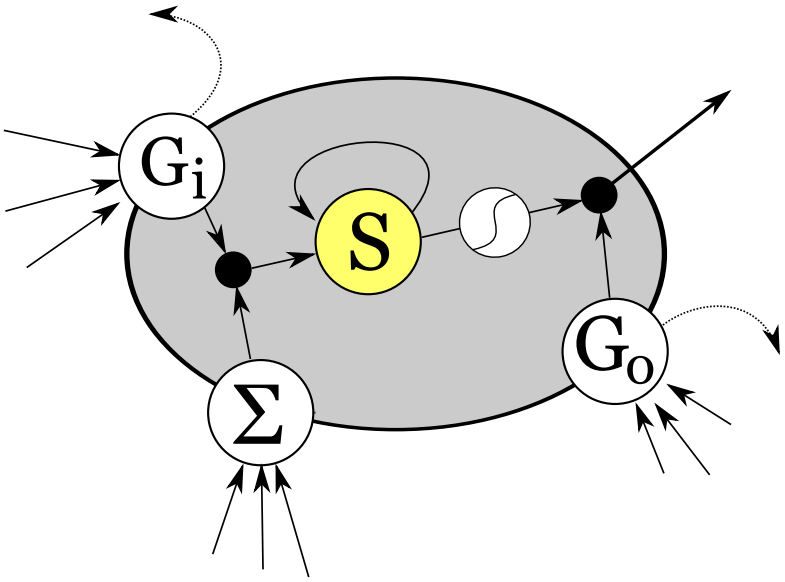

Abbildung 3: Schema einer Memory-Cell der Größe 1. Jeder fortführende Pfeil kann dabei als Eingabe für eine Memory-Cell (auch dieselbe) verstanden werden. Nach der selben Überlegung können die hinführenden Pfeile von jeder Memory-Cell bzw. der Eingabeschicht kommen. Die Ausgabeschicht erhält nur die Ausgabe der Zustände als Input (dicker Pfeil).

Der Wert des Zustands im jeweils nächsten Zeitschritt wird additiv aus seinem vorherigen Wert und der Netzeingabe der Memory-Cell (Σ) bestimmt. Die Netzeingabe wird dabei von der Ausgabe des Input-Gates multiplikativ skaliert. Die Ausgabe der Zelle entspricht der Ausgabe des Zustands, multiplikativ skaliert mit der Ausgabe des Output-Gates.

Für jede der Einheiten in einer Memory-Cell (G_i, G_o, Σ und S) werden eigene Ausgabefunktionen definiert. Ein LSTM kann i.A. aus mehreren solcher Memory-Cells bestehen sowie aus Memory-Cells unterschiedlicher Größe. Im letzten Fall teilen sich mehrere Zustände dieselben Gates und dieselbe Netzeingabe. Der Einfachheit werden im folgenden immer Memory-Cells der Größe 1 angenommen.

Folgende Gleichungen beschreiben die Vorwärtsaktivierung der Memory-Cell c_j:

$$s_{c_j}(t) = s_{c_j}(t-1) + f_{in_j}(net_{in_j}(t)) \cdot g(net_{c_j}(t))$$
$$y_{c_j}(t) = f_{out_j}(net_{out_j}(t)) \cdot h(s_{c_j}(t))$$

Dabei ist f_{in_j}, f_{out_j}, g und h die Ausgabefunktion des Input-Gates, des Output-Gates, der Netzeingabe (Σ weiter oben) bzw. des Zustands. $net_i(t)$ ist die (Skalarprodukt-) Aktivierung der Zelle i. Die definition der Netzeingabe jeder Einheit bleibt dabei dem Benutzer überlassen. Im Allgemeinen kann jede Ausgabe einer Einheit jeder anderen Einheit zugeführt werden.

Eine Memory-Cell ist in der Lage durch das Input-Gate zu lernen, wann die im Zustand gespeicherte Information von unwichtigen Eingaben zu schützen ist. Durch das Output-Gate lernt sie, wann sie ihren Einfluss auf andere Zellen oder die Ausgabeschicht reduzieren soll. Dies ermöglicht es dem LSTM wichtige Informationen über lange Zeitabstände hinweg zu lernen und zeitlich zu generalisieren[2]. [Hochreiter und Schmidhuber, 1997], [Hochreiter, 1998]

Bei der hier gezeigten Architektur handelt es sich um die ursprüngliche Version des LSTM. Weitere Forschung im laufe der Zeit hat Nachteile offengelegt, die durch neue Einheiten und Verbindungen, wie in Abschnitt 5.1 gezeigt, behoben wurden. So zeigte das LSTM seine Vorteile hauptsächlich im Bereich der diskreten, rauschfreien Sequenzanalyse und -erkennung. [Hochreiter und Schmidhuber, 1997], [Gers und Schmidhuber, 2000]

Mit der weiterentwicklung des LSTM war es auch in der Lage aus der Länge von Zeitintervallen Information zu extrahieren, was es für die Klassifizerung von Musik und Rhytmus nützlich machte. Auch zur Spracherkennung wurde das LSTM erfolgreich Eingesetzt. [Gers et al., 2000],[Eck und Schmidhuber, 2002], [Graves, 2008]

Als neuroevolviertes RNN (siehe Abschnitt 5.2) fand das LSTM Verwendung zur Vorhersage von Zeitreihen. Diese waren zuerst auf synthetische Zeitreihen wie die Mackey-Glass-Reihe oder überlagerte Sinusschwingungen beschränkt. Dort zeigte es gute Ergebnisse, konnte das ESN aber nur bezüglich der Sinusschwingungen übertreffen. [Wierstra et al., 2005] Zuletzt wurden mithilfe des LSTM Trajektorien eines Roboters für eine Schleifenbewegung generiert. Dabei erhielt das LSTM die aktuelle Position als Eingabe und gab die vorhergesagte Positionsänderung aus. [Mayer et al., 2006]

[2] Die Eigenschaft „zeitliche Generalisierung" ist in [Hochreiter, 1998] wie folgt definiert: *Ein trainiertes Netz kann einen Zielwert der Testmenge approximieren, der einen in der Trainingsmenge nicht vorkommenden zeitlichen Abstand zu den für seine Berechnung relevanten Eingaben hat. (...)*

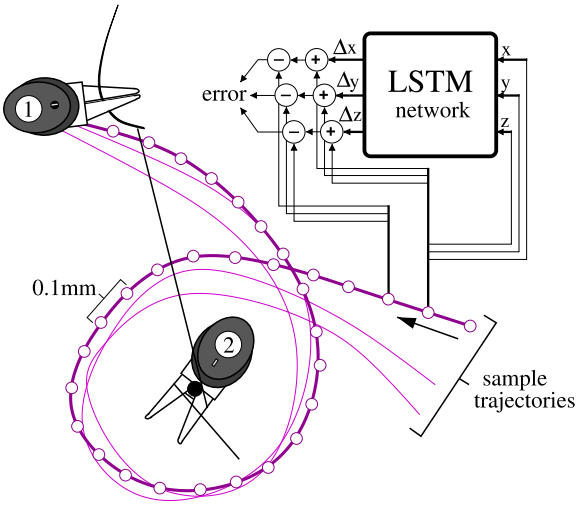

Abbildung 4: Training eines LSTMs zur Generierung von Trajektorien für eine Schleife. Die violetten Linien sind Beispieltrajektorien. (aus [Mayer et al., 2006])

4 Verschwindender Fehlergradient und das „konstante Fehlerkarussell"

Der verschwindende Fehlergradient ist ein Phänomen, welches die Lernfähigkeit von RNNs bei gradientenbasierten Trainingsmethoden erheblich einschränkt. Das Lernen von Abhängigkeiten über große zeitliche Abstände wird dadurch erschwert oder gar unmöglich.

4.1 Beschreibung des Problems

Ein RNN benutzt zu jedem Zeitschritt die selben Gewichte um die Ausgabe zu berechnen. Der innere Zustand ändert sich jedoch im Laufe der Zeit. Folglich haben Gewichte in unterschiedlichen Zeitpunkten auch verschiedenen Einfluss auf den Fehlergradienten. Der Fehlergradient an einem Gewicht w_{ij} zum Zeitpunkt t ist die Summe der Fehlergradienten zu allen Zeitpunkten $\tau \leq t$ (siehe Abschnitt 2):

$$\frac{\partial F(t)}{\partial w_{ij}} = \sum_{\tau \leq t} \frac{\partial F(t)}{\partial w_{ij}(t-\tau)}$$

Es lässt sich nun eine Skalierung des Fehlergradienten $\frac{\partial F(t)}{\partial w_{ij}(t-\tau)}$ nachweisen, wobei der Skalierungsfaktor exponentiell vom zeitlichen Abstand τ abhängt[3].

[3] Eine genaue Herleitung findet sich in [Hochreiter, 1998]. In [Bengio et al., 1994] findet sich eine Herleitung auf Basis systemtheoretischer Betrachtungen.

Für den Fall, dass der Faktor mit τ exponentiell zunimmt, handelt es sich bei dem Netz um ein instabiles Netz. Es äußert sich durch oszillierende Gewichte und chaotisches Verhalten. Dies kann jedoch durch die Wahl der Anfangsgewichte und durch die Trainingsmethode i.A. verhindert werden.

Der Fall der exponentiellen Abnahme hingegen ist die Ursache für das angesprochene Phänomen: Gradienteninformation von Zeitschritten die weit zurückliegen sind klein im Vergleich zu Kurzzeitinformationen oder haben effektiv überhaupt keinen Einfluss mehr. Der Fehlergradient „verschwindet".

4.2 Der Lösungsansatz des LSTM

Aus der Analyse des Verschwindenden Gradienten ergibt sich die Forderung nach einem konstanten Fehlerrückfluss an den Einheiten eines RNNs. Dieser Fehlerrückfluss ist nach [Hochreiter, 1998] für eine Einheit j definiert als

$$\vartheta_j(t) = f_j'(net_j(t)) \cdot \vartheta_j(t+1) \cdot w_{jj} \quad [4].$$

Um diesen nun über die Zeit konstant zu halten, muss

$$\frac{\vartheta_j(t)}{\vartheta_j(t+1)} = f_j'(net_j(t)) \cdot w_{jj} = 1$$

gelten. Nach Integration erhält man:

$$f_j(net_j(t)) \cdot w_{jj} = net_j(t) + C$$

Wählt man $C = 0$, $f_j(x) = x$ als Identität, und das Gewicht der rekurrenten Verbindung $w_{jj} = 1$ ist die Forderung für beliebige $net_j(t)$ gewährleistet. Eine Einheit j mit w_{jj} bildet ein „Fehlerkarussell" und ermöglicht einen konstanten Fehlerrückfluss. Da durch die sonstigen hin- und wegführenden Gewichte jedoch widersprüchliche Fehlersignale entstehen würden, wurden die in Abschnitt 3.3 erwähnten Gate-Einheiten eingeführt.

5 Neuroevolution für LSTM - EVOLINO

Die Abkürzung *EVOLINO* steht für *EVOlution of systems with LINear Outputs*. Konzipiert war es als allgemeines Framework um dynamische Systeme (zu welchen RNNs gehören) mithilfe von

[4] Hier wird nur der lokale Fehlerrückfluss zur Einheit selbst betrachtet

genetischen Algorithmen zu modellieren. Die Ausgabeschicht wird dabei genau wie die ebenfalls lineare Ausgabeschicht im ESN per linearer Regression eingestellt. Obwohl wie gesagt als allgemeines Framework gedacht, wurde es bisher lediglich auf das in dieser Arbeit vorgestellte LSTM angewandt[5]. Dieses wurde von der ersten Erwähnung bis zur Vorstellung des EVOLINO stetig verbessert. Inwiefern zeigt der nächste Abschnitt. Die übrigen Abschnitte erläutern, wie neuroevolution im EVOLINO konkret auf das LSTM angewendet wurde und vergleichen die beiden Architekturen direkt.

5.1 Vebesserung des LSTM

Das ursprüngliche LSTM wurde auf dem Weg zur Neuroevolution um verschiedene architekturspezifische Details erweitert. Diese Erweiterungen verbesserten die Leistung des LSTM und erweiterten die Lernfähigkeit.

Eine Verbesserung ermöglichte das sogenannte „Forget-Gate": Eine zusätzliche Gate-Einheit, welche die direkte Rückführung des Zustands skaliert. Die neue Gleichung für die aktualisierung des Zustands lautet dementsprechend:

$$s_{c_j}(t) = f_{forget_j}(net_{forget_j}(t)) \cdot s_{c_j}(t-1) + f_{in_j}(net_{in_j}(t)) \cdot g(net_{c_j}(t))$$

Diese Skalierung ermöglicht es dem LSTM auch kontinuierliche Zeitreihen zu verarbeiten, die nicht von vornherein in Trainingsdatensätze zerlegt wurden. Dies wird erreicht, da eine Memory-Cell nun in der Lage ist, seinen Zustand vollständig oder teilweise zurückzusetzen. Ohne Forget-Gates würden Zustände im LSTM bei kontinuierlicher Verarbeitung von Zeitreihen über alle Grenzen wachsen. Das Forget-Gate wird analog zu den anderen Gates trainiert. [Gers et al., 2000]

Weiterhin wurden sogenannte „Peephole-Verbindungen" eingeführt: Dabei handelt es sich um Verbindungen vom Zustand der Memory-Cell zu den Gates. Ein LSTM das über solche Peephole-Verbindungen verfügt ist in der Lage, Information aus Zeitabständen von Eingaben zu extrahieren (das LSTM „lernt zu zählen"). Um die Funktion der Gates zu erhalten, wird über diese Peephole-Verbindungen kein Fehlersignal zum Zustand zurückpropagiert. Es werden lediglich die Gewichte eingestellt. [Gers und Schmidhuber, 2000]

Für Zeitreihenprädiktion scheint es sich für nützlich erwiesen zu haben, ein „Ausgabe-Peephole" zu verwenden, welches von den Zuständen direkt zur Ausgabeschicht führt. Die für die Versu-

[5] Im folgenden ist daher mit EVOLINO immer die Kombination EVOLINO/LSTM gemeint.

che mit neuroevolution verwendete Architektur ist in Abb. 5 gezeigt. [Wierstra et al., 2005], [Schmidhuber et al., 2007]

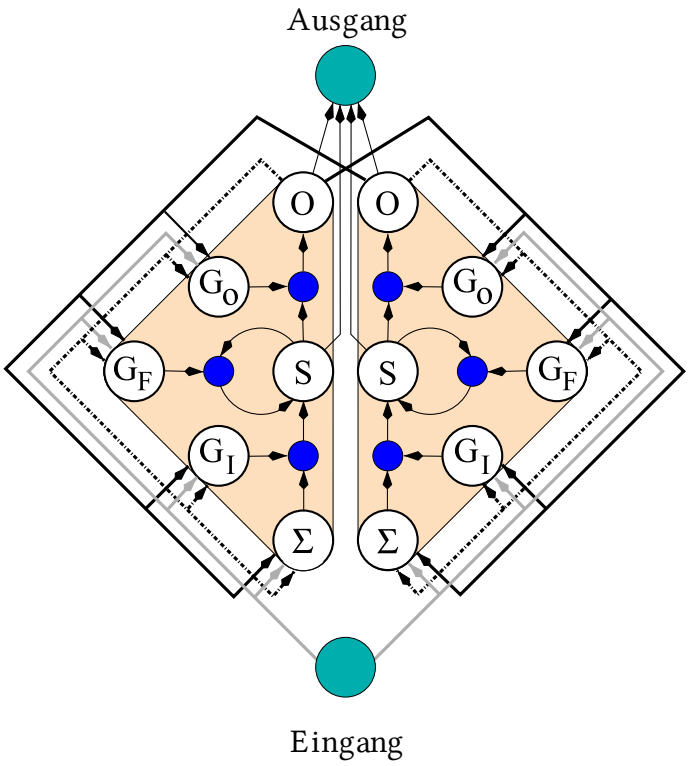

Abbildung 5: Ein LSTM mit zwei Memory-Cells und einem Ein- und Ausgang, wie es zur neuroevolvierten Zeitreihenprädiktion verwendet wird. Hinzugekommen sind Forget-Gates und Ausgabe-Peepholes. Peephole-Verbindungen zu den Gates sind nicht vorhanden, da ihre Funktionalität in der Anwendung nicht benötigt wird. Ein weiterer Unterschied ist die direkte Skalierung des Zustands auf dem Weg zur Ausgabefunktion (Hier O, in Abb. 3 schematisch angedeutet) durch das Output-Gate. (aus [Wierstra et al., 2005])

5.2 Neuroevolution

Das EVOLINO nutzt zur evolvierung des LSTM ein *Enforced Subpopulations* (ESP) genanntes Konzept. Wie in Abschnitt 2.4 erwähnt gibt es zahlreiche Möglichkeiten genetische Algorithmen auf Neuronale Netze anzuwenden. ESP ist nun eine Methode um die Gewichte eines Neuronalen Netzes zu evolvieren. Dabei wird jedes Neuron eines Netzes von einer eigenen *Subpopulation* von Individuen repräsentiert (im Gegensatz zur Repräsentation des ganzen Netzes mit nur einer Population). Vorteil dabei ist, dass nun diese Neuronen schneller eine Spezialisierung im Netz

ausbilden können und diese auch behalten. [Gomez, 2003] Im Fall des LSTMs wird nun jede einzelne Memory-Cell von einer solchen Subpopulation kodiert. Abbildung 6 veranschaulicht dies.

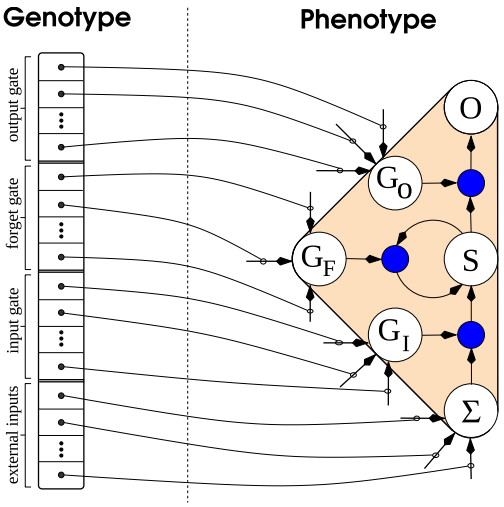

Abbildung 6: Jede Subpopulation repräsentiert eine Memory-Cell im von EVOLINO trainierten LSTM. Dabei werden die Gewichte nur mittels Mutation evolviert. In der Realisierung wird hierzu dem Gewichtsvektor ein Zufallsvektor hinzuaddiert. (aus [Wierstra et al., 2005])

Der Trainingsablauf lässt sich grob in zwei Schritte einteilen (für eine detailiertere Ausführung des evolvierenden Teils siehe [Schmidhuber et al., 2007]):

1. Ein neu instanziiertes Netz wird auf das Problem angesetzt, bis genug Datensätze vorhanden sind um die Ausgabeschicht per Regression einzustellen. Dieser Schritt ist identisch mit dem Training eines ESN

2. Das nun trainierte Netz wird wieder auf das Problem angesetzt. Diesmal allerdings wird das LSTM wie erwähnt in Subpopulationen aufgeteilt und evolviert. Zur Evaluation wird ein komplettes Netz mit je einem Individuum aus jeder Subpopulation zusammengestellt. Als Maß für die Fitness dient der Fehler, der an der Ausgabeschicht auftritt.

Als genetischer Operator wird dabei lediglich Mutation verwendet. Die Begründung hierfür ist, dass Neuroevolution das Feintuning der Prädiktionsaufgabe übernimmt. Es entspricht faktisch der stochastischen Suche in der engeren Nähe um die jeweils aktuell gefundene Lösung. Gradientenabstieg, für welches das LSTM ursprünglich entwickelt wurde, kommt nicht mehr zum Einsatz.

5.3 Im Vergleich mit dem ESN

Mit der linearen Ausgabeschicht des EVOLINO basiert dieses auf der selben Grundidee wie das ESN: Dynamiken eines RNNs auslesen und kombinieren statt die gewünschte Dynamik direkt zu modellieren[6]. Das EVOLINO kann damit zu den Vertretern des Reservoir Computing gezählt werden. Im Unterschied zur ursprünglichen herangehensweise, das Reservoir zufällig zu instanziieren und im weiteren Verlauf untrainiert zu lassen, wird beim EVOLINO das Reservoir eben mittels neuroevolution optimiert. Bei der Verwendung von LSTM ist die Grundstruktur des Reservoirs dabei ein RNN, das sich bei Problemen mit langen Zeitabhängigkeiten bereits bewährt hat.

Wenn das ESN in puncto Genauigkeit auch besser abschneidet als EVOLINO, so wartet letzteres hingegen mit einem breiterem Anwendungsspektrum auf. So ist das EVOLINO auch in der Lage sehr effizient diskrete Probleme wie Paritätsprüfung oder das Lernen von Grammatiken zu lösen. Das ESN scheint hierfür weniger gut geeignet zu sein.

Fazit

Reale Bewegungsdaten wurden in der recherchierten Literatur lediglich ein einziges Mal prädiziert ([Hellbach et al., 2008]). Dabei haben sich ESN als durchaus nützlich erwiesen. Wie zu Beginn erwähnt, stellen RNNs allgemein ein wichtiges Werkzeug im Bereich der Zeitreihenvorhersage dar. Das ESN oder Reservoir Computing haben der Verwendung von RNNs in diesem Bereich nun zusätzlich einen deutlichen Aufschwung beschert. EVOLINO verbindet nun Reservoir Computing mit neuroevolution und einer „besonderen Stärke" für diskrete Probleme und lange Zeitabhängigkeiten (LSTM).

Diese Aspekte lassen EVOLINO zur Prädiktion von Bewegungsdaten durchaus vielversprechend erscheinen. Letzte Sicherheit würde natürlich nur die tatsächliche Anwendung auf reale Bewegungsdaten bringen. Im Zuge dessen wäre vielleicht auch interessant, die Anwendbarkeit des EVOLINO auf andere RNNs zu Untersuchen oder die Kombination mit dem Gradientenabstieg für das LSTM zu versuchen.

[6] *„Die Intuition ist, dass es leichter ist eine ausreichend gute Basis zu finden als zu versuchen ein Netz zu finden das das System exakt selbst modelliert."* [Schmidhuber et al., 2007]

Literatur

[Bengio et al., 1994] Bengio, Y., P. Simard und P. Frasconi (1994). *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 5(2):157–166.

[Eck und Schmidhuber, 2002] Eck, D. und J. Schmidhuber (2002). *Finding temporal structure in music: blues improvisation with LSTM recurrent networks*. S. 747–756.

[Floreano et al., 2008] Floreano, Dario, P. Dürr und C. Mattiussi (2008). *Neuroevolution: from architectures to learning*. Evolutionary Intelligence, 1(1):47–62.

[Gers und Schmidhuber, 2000] Gers, F. A. und J. Schmidhuber (2000). *Recurrent nets that time and count*. S. 189–194.

[Gers et al., 2000] Gers, F. A., J. Schmidhuber und F. Cummins (2000). *Learning to forget: Continual prediction with LSTM*. Neural Computation, 12(10):2451–2471.

[Gomez, 2003] Gomez, Faustino John (2003). *Robust Non-linear Control through Neuroevolution*. Technischer Bericht, Artificial Intelligence Laboratory, The University of Texas at Austin.

[Graves, 2008] Graves, A. (2008). *Supervised sequence labelling with recurrent neural networks*. Doktorarbeit, Technische Universität München.

[Haykin, 2009] Haykin, S. S. (2009). *Neural Networks and Learning Machines*. Prentice Hall.

[Hellbach et al., 2008] Hellbach, S., S. Strauss, J. P. Eggert, E. Körner und H. M. Gross (2008). *Echo State Networks for Online Prediction of Movement Data - Comparing Investigations*. ICANN 2008, Part I, LNCS, 5163:710–719.

[Hochreiter, 1998] Hochreiter, Josef (1998). *Generalisierung bei Neuronalen Netzen geringer Komplexität*. Doktorarbeit, TU München.

[Hochreiter und Schmidhuber, 1997] Hochreiter, S. und J. Schmidhuber (1997). *Long short-term memory*. Neural Computation, 9(8):1735–1780.

[Ishu et al., 2004] Ishu, K., T. van der Zant, V. Becanovic und P. Ploger (2004). *Identification of motion with echo state network*. OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04, 3.

[Jaeger, 2002] Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*.

[Jaeger, 2003] Jaeger, H. (2003). *Adaptive Nonlinear System Identification with Echo State Networks*. Advances in Neural Information Processing Systems, 8:9.

[Lukosevicius und Jaeger, 2009] Lukosevicius, M. und H. Jaeger (2009). *Reservoir Computing Approaches to Recurrent Neural Network Training*. Computer Science Review.

[Mandic und Chambers, 2000] Mandic, D. P. und J. A. Chambers (2000). *Recurrent Neural Networks for Prediction*. Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications, and Control. Wiley.

[Mayer et al., 2006] Mayer, H., F. Gomez, D. Wierstra, I. Nagy, A. Knoll und J. Schmidhuber (2006). *A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks*. Advanced Robotics, 22, 13(14):543–548.

[Mirikitani und Ouarbya, 2009] Mirikitani, Derrick Takeshi und L. Ouarbya (2009). *Modelling D_{st} with Recurrent EM Neural Networks*. ICANN Part I, LNCS, 5768:975–984.

[Namikawa und Tani, 2008] Namikawa, J. und J. Tani (2008). *A model for learning to segment temporal sequences, utilizing a mixture of RNN experts together with adaptive variance*. Neural Networks, 21(10):1466–1475.

[Pohlheim, 1999] Pohlheim, Hartmut (1999). *Evolutionäre Algorithmen*. Springer Berlin.

[Projekt PHOCUS, 2010] Projekt PHOCUS (2010). http://cordis.europa.eu/fetch?CALLER=DE_NEWS&ACTION=D&DOC=1&CAT=NEWS&QUERY=012800c3303f:ca18:6e7c3046&RCN=31811.

[Puskorius und Feldkamp, 1994] Puskorius, G. V. und L. A. Feldkamp (1994). *Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks*. IEEE Transactions on Neural Networks, 5(2):279–297.

[Qian-Li et al., 2008] Qian-Li, M., Z. Qi-Lun, P. Hong, Z. Tan-Wei und Q. Jiang-Wei (2008). *Multi-step-prediction of chaotic time series based on co-evolutionary recurrent neural network*. Chinese Physics B, 17(2):536.

[Reservoir Computing, 2010] Reservoir Computing (2010). *reservoir-computing.org*. http://reservoir-computing.org.

[Schmidhuber et al., 2007] Schmidhuber, J., D. Wierstra, M. Gagliolo und F. Gomez (2007). *Training recurrent networks by evolino*. Neural computation, 19(3):757–779.

[Schmidhuber, 1993] Schmidhuber, Jürgen (1993). *Netzwerkarchitekturen, Zielfunktionen und Kettenregel*. TU München.

[Sumpter und Bulpitt, 2000] Sumpter, N. und A. Bulpitt (2000). *Learning spatio-temporal patterns for predicting object behaviour*. Image and Vision Computing, 18(9):697–704.

[Unkelbach et al., 2009] Unkelbach, J., S. Yi und J. Schmidhuber (2009). *An EM Based Training Algorithm for Recurrent Neural Networks*. ICANN 2009, Part I, LNCS, 5768:964–974.

[Webb, 2008] Webb, R. (2008). *Time Series Prediction with Evolved, Composite Echo State Networks*. Simulated Evolution and Learning, S. 555–564.

[Wierstra et al., 2005] Wierstra, D., F. J. Gomez und J. Schmidhuber (2005). *Modeling systems with internal state using evolino*. S. 1802.

[Yao, 1993] Yao, Xin (1993). *Evolutionary Artificial Neural Networks*. International Journal of Neural Systems, 4(3):203–222.

[Yao, 1999] Yao, Xin (1999). *Evolving Artificial Neural Networks*. Proceedings of the IEEE, 87(9):1423–1447.

[Zell, 1994] Zell, Andreas (1994). *Simulation neuronaler Netze*. Oldenbourg, 4. Aufl.