GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Fakultät für Physik

Bericht

Spezialisierungspraktikum

angefertigt von

Roland Simon Zimmermann

aus Recklinghausen

am Max-Planck-Institut für Dynamik und Selbstorganisation

Bearbeitungszeit: 1. März 2017 bis 29. März 2017

Erstgutachter/in: Prof. Dr. Ulrich Parlitz

Inhaltsverzeichnis

Nomenklatur

Symbol	Bedeutung
N	Anzahl der Einheiten im Reservoir $\mathbf{W}$
α	Verlustrate des *Leaky Integrators*
β	Parameter für den Lernvorgang mittels *Tikhonov Regularisierung.*
$\rho(\mathbf{W}))$	Spektralradius von $\mathbf{W}$
$\sigma(\mathbf{W})$	Singulärwert von $\mathbf{W}$
ν_{max}	Maximalwert des hinzugefügten Rauschens $\nu(n)$ für die Erhöhung der Stabilität.
$[\cdot\,;\,\cdot]$	Vertikales Aneinanderfügen von Vektoren/Matrizen

1 Einleitung

Mittels *Neuronaler Netzwerke* konnten in den vergangenen Jahren viele Probleme des *Machine Learnings* und der datenbasierten Ereignisvorhersage gelöst werden. Hierbei hat sich allerdings die Vorhersage von Zeitserien lange Zeit als problematisch erwiesen - dies änderte sich erst, als rekurrente Netzwerke vermehrt genutzt wurden. Eine Form dieser Netzwerke sind die Echo State Networks aus dem Bereich des *Reservoir Computings*. Sie stellen einen vereinfachten Ansatz dar, welche teilweise bemerkenswerte Ergebnisse bei der Analyse und Vorhersage von Zeitserien liefern kann.

Dieser Bericht gibt eine Übersicht über die im Spezialisierungspraktikum gewonnen Erkenntnisse bezüglich Echo State Networks. Hierbei wurden zuerst die theoretischen Grundlagen betrachtet und die dabei gewonnenen Informationen anschließend auf zwei Anwendungsbeispiele bezogen.

Alle Programmierbeispiele während des Praktikums wurden in Python mittels numpy und scipy erstellt. Die relevanten Auszüge dieser werden dem Protokoll in digitaler Form beigelegt.

2 Überblick über Neural Networks

In den letzten Jahren hat die Technik der Neuronalen Netzwerke erneut stark an Popularität gewonnen. Dies liegt zum einen an der gestiegenen verfügbaren Rechenleistung und zum anderen an der Entwicklung hierfür notwendiger Algorithmen.
Allgemein lassen sich diese Netze in zwei große Gruppen aufteilen: die der Feed Forward Neural Networks und die der Recurrent Neural Networks, welche im Folgenden als FFNN respektive RNN bezeichnet werden.

Ein FFNN besteht aus mehreren Ebenen, welche jeweils aus verschiedenen nicht-linearen Einheiten zusammengesetzt sind. Die erste dieser Ebenen wird zur Eingabe und die letzte zur Ausgabe eines Signals genutzt. Die Einheiten zweier benachbarter Ebenen sind mit individuellen Gewichten vollständig in Richtung der Ausgabe verbunden. Dies bedeutet, dass jedes Einheit x_i^n sein Signal an alle Einheiten der folgenden Ebene x_j^{n+1} mit einem individuellen Gewicht $w_{i\rightarrow j}^n$ weitergibt. Zwischen den Einheiten innerhalb einer Ebene bestehen keinerlei Verbindungen. Damit ein solches Netzwerk Vorhersagen treffen kann, müssen die Gewichte in einem Trainingsvorgang angepasst werden. Dies wurde durch die Entwicklung des Backpropagation-Algorithmus stark vereinfacht. Hierbei werden die Gewichte so angepasst, dass eine Kostenfunktion minimiert wird [1, S. 225-290].

Ein RNN hat einen ähnlichen Aufbau, doch hier können alle Einheiten an alle anderen Einheiten Signale weitergeben und von diesen erhalten. Dies kann die Vorhersage in bestimmten Anwendungsbeispielen wie der Text- und Sprachanalyse verbessern. Ein Nachteil ist, dass zum Trainieren nicht mehr der einfachere Backpropagation-Algorithmus Algorithmus genutzt werden kann, sondern eine für RNNs abgewandelte Form genutzt werden muss. Dafür werden die verschiedenen Zustände die das RNN im Laufe der Signal-Propagation annimmt nacheinander betrachtet und auf diese zeitliche Entwicklung anschließend der Backpropagation-Algorithmus angewendet. Diese Methode ist unter dem Namen Backpropagation through Time (BTT) bekannt. Sie ist zum einen rechenaufwendiger und zum anderen auch instabiler, da das Verschwindenden des Gradienten der Kostenfunktion deutlich wahrscheinlicher als bei der gewöhnlichen Backpropagation ist [10].

3 Echo State Networks

3.1 Überblick

Um die (Leistungs)Probleme der RNN zu umgehen, wurden als mögliche Lösung die Echo State Networks von H. Jäger vorgeschlagen [2]. Etwa zeitgleich wurde von W. Maas das Modell der *Liquid State Machines* vorgeschlagen. In diesem Modell steht der biologische Hintergrund im Fokus, doch sind die Ergebnisse denen der Echo State Networks sehr ähnlich [8].

3.2 Aufbau

Ein ESN ist eine Spezialform eines RNNs. Hierbei wird eine auf dem ersten Blick eigenartige Entscheidung getroffen: Während des gesamten Trainingsvorganges werden die Verbindungen der einzelnen Einheiten größtenteils nicht verändert. Es wird versucht durch das *Echo* der vorherigen Signale, welche noch im Netzwerk gespeichert sind, diese Signale zu rekonstruieren - hieraus ergibt sich auch der Name [6]. Im Folgenden wird der Aufbau und anschließend die Funktionsweise eines solchen Netzwerkes nach [5] beschrieben.

Allgemein bildet das Netzwerk S ein zeitliches Signal $\vec{u}(n) \in \mathbb{R}^{N_u}$ auf eine zeitlich variable Ausgabe $\vec{y}(n) \in \mathbb{R}^{N_y}$ für die Zeiten $n = 1, ..., T$ ab. Zudem besitzt das System ein sogenanntes Reservoir aus N nicht-linearen Einheiten. Der innere Zustand des Netzwerkes wird durch diese Einheiten beschrieben und als $s(n) \in \mathbb{R}^N$ bezeichnet.
Die Verbindungen der inneren Einheiten untereinander werden durch die Gewichtsmatrix $\mathbf{W} \in \mathbb{R}^{N\times N}$ beschrieben. Das Eingangssignal wird zusammen mit einem *Bias* $b_{in} \in \mathbb{R}$ durch die Matrix $\mathbf{W}_{in} \in \mathbb{R}^{N\times(N_u+1)}$ auf die inneren Einheiten weitergeleitet.

Die zeitliche Entwicklung der inneren Zustände berechnet sich nach der Vorschrift

$$\vec{s}(n) = (1-\alpha)\cdot\vec{x}(n-1) + \alpha\cdot f_{in}\left(\mathbf{W}_{in}[b_{in};\vec{u}(n)] + \mathbf{W}\vec{x}(n-1)\right), \qquad (1)$$

wobei f_{in} eine beliebige (meistens *sigmoid*-förmige) Transferfunktion ist, und $[\cdot\,;\,\cdot]$ das vertikale Aneinanderfügen von Vektoren beziehungsweise Matrizen bezeichnet. Für diese Zustandsgleichung wurde das Modell eines *Leaky Integrator Neurons* genutzt, wobei $\alpha \in (0,1]$ die Verlustrate beschreibt. Für $\alpha = 1$ ergibt sich als Spezialfall ein gewöhnliches

Neuron

$$\vec{s}(n) = f_{in}\left(\mathbf{W}_{in}[b_{in}; \vec{u}(n)] + \mathbf{W}\vec{x}(n-1)\right). \tag{2}$$

Da für manche Anwendungsfälle auch eine direkte Rückkopplung wünschenswert ist, kann das System noch um eine Ausgabe-Rückkopplung erweitert werden. Diese verbindet die Ausgabe erneut mit den inneren Einheiten durch die Matrix $\mathbf{W}_{fb} \in \mathbb{R}^{N \times N_y}$. Somit ergibt sich

$$\vec{s}(n) = (1-\alpha) \cdot \vec{x}(n-1)\alpha \cdot f_{in}\left(\mathbf{W}_{in}[b_{in}; \vec{u}(n)] + \mathbf{W}\vec{x}(n-1) + \mathbf{W}_{fb}\vec{y}(n)\right) \tag{3}$$

als Zustandsgleichung.

An Hand der inneren Zustände lassen sich nun noch die sogenannten erweiterten inneren Zustände $x(n) = [b_{out}; \vec{s}(n); \vec{u}(n)] \in \mathbb{R}^{1+N_u+N}$ definieren, wobei b_{out} ein *Bias* für die Ausgabe darstellt.

Aus diesen erweiterten inneren Zuständen kann nun die Ausgabe $\vec{y}(n)$ konstruiert werden. Dies kann entweder im Sinne einer Linearkombination durch die Ausgangsmatrix $\mathbf{W}_{out} \in \mathbb{R}^{(1+N_u+N) \times N_y}$ oder durch andere nicht lineare Klassifizierer wie beispielsweise einer Support Vector Machine (SVM) durchgeführt werden. Im Folgenden wird nur der Fall einer Linearkombination betrachtet, da sich für die anderen Methoden ein analoges Verfahren ergibt. In diesem Fall berechnet sich die Ausgabe mittels

$$\vec{y}(n) = f_{out}\left(\mathbf{W}_{out}\vec{x}(n) = \mathbf{W}_{out}[b_{out}; \vec{s}(n); \vec{u}(n)]\right), \tag{4}$$

wobei f_{out} die Transferfunktion der Ausgabe ist.

Während die Matrix $\mathbf{W}_{out}$ durch den Trainingsvorgang bestimmt wird, werden die Matrizen $\mathbf{W}_{in}$ und $\mathbf{W}$ a priori generiert und festgelegt. Hierbei hat sich für das Generieren der Eingangsmatrix eine zufällige Anordnung von zufälligen Gleitkommazahlen zwischen -1.0 und 1.0 als geschickt herausgestellt. Falls ein Feedback gewünscht ist, also Gleichung (3) genutzt wird, wird $\mathbf{W}_{fb}$ gleichartig konstruiert. Auf das Generieren der inneren Matrix $\mathbf{W}$ wird in Abschnitt 3.4 genauer eingegangen.

3.3 Trainingsvorgang

Nachdem der Aufbau des Netzwerkes beschrieben ist, ergibt sich nun die Frage, wie der Trainingsvorgang durchgeführt wird.
Hierfür wird für die Zeiten $n = 0, ..., T_0$ das ESN mit dem Signal $\vec{u}(n)$ betrieben, wobei T_0 die *transiente Zeit* beschreibt. Hierdurch soll das System aus seinem zufällig gewähltem Anfangszustand in einen charakteristischen Zustand übergehen. Anschließend wird das System für Zeiten $n < T$ weiter betrieben und die erweiterten Zustände $\vec{x}(n)$ als Spalten in der *Zustandsmatrix* $\mathbf{X} \in \mathbb{R}^{(1+N_u+N)\times T}$ gesammelt. Analog dazu werden die gewünschten Ausgaben $\vec{y}(n)$ nach dem Anwenden der Inversen f_{out}^{-1} der Ausgabe-Transferfunktion f_{out} auch als Spalten in der *Ausgabematrix* $Y \in \mathbb{R}^{N_y \times T}$ gesammelt. Nun wird eine Lösung der Gleichung

$$\mathbf{Y} = \mathbf{W}_{out}\mathbf{X} \tag{5}$$

für $\mathbf{W}_{out}$ gesucht. Hierfür stehen mehrere Verfahren zur Verfügung, von denen zwei prominente erwähnt sein sollen. Zum einen kann die Lösung durch eine *Tikhonov Regularisierung* mittels der Regularisierung $\beta \cdot ||\vec{W}_{out,i}||^2$ der Gewichtsmatrix mit der Konstante β erhalten werden. Hierbei steht $\vec{W}_{out,i}$ für die jeweils i-te Zeile der Gewichtsmatrix. Das Verfahren

$$\mathbf{W}_{out} = \mathbf{Y}\mathbf{X}^T \left(\mathbf{X}\mathbf{X}^T + \beta I\right)^{-1} \tag{6}$$

ist sehr leistungsstark, aber auch teilweise numerisch instabil. Bei geeigneter Wahl von β können die besten Ergebnisse hinsichtlich der Genauigkeit der Vorsage erzielt werden [6].

Zum anderen kann zur Lösung die *Moore-Penrose-Pseudoinverse* $\mathbf{X}'$ genutzt werden, sodass für die Ausgabematrix

$$\mathbf{W}_{out} = \mathbf{Y}\mathbf{X}' \tag{7}$$

folgt. Dieses Verfahren ist zwar sehr rechenaufwendig aber dafür numerisch stabil [4, 6]. Nichts desto trotz, kann allerdings auf Grund des Fehlens einer Regularisierung leicht der Effekt des OVERFITTINGS auftreten. Dieser kann umgangen werden, indem in der Zustandsgleichung (1) beziehungsweise (3) eine leichte normalverteilte Störung $\vec{\nu}(n)$ der Größenordnung 1×10^{-1} bis 1×10^{-5} addiert wird. Dieser Ansatz beruht auf Empirie, da eine mathematische Begründung hierfür noch nicht vollständig gelungen ist [2, 6].

Zusammenfassend ergibt sich somit der folgende Funktionsablauf für die Anwendung eines ESN:

1. Zufälliges Generieren der Matrizen $\mathbf{W}_{in}, \mathbf{W}_{fb}$ und Konstruktion der Matrix $\mathbf{W}$
2. Einspeisen des Signals $u(n)$ und Konstruktion der Zustandsmatrix $\mathbf{X}$ und der Ausgabematrix $\mathbf{Y}$
3. Berechnung der Ausgabematrix $\mathbf{W}_{out}$
4. Einspeisen des Signals $u(n)$ für Vorhersagen des Signales $y(n)$ für $n > T$

3.4 Theoretischer Hintergrund

Um die mathematischen Eigenschaften beschreiben zu können, sind zuerst zwei Definitionen nötig [11].

Definition 1 (Kompatibler Zustand)**.** *Sei $S : X \times U \to X$ ein* ESN *mit der Zustandsgleichung $\vec{x}_{n+1} = F(\vec{x}_n, \vec{u}_{n+1})$. Eine Folge von Zuständen $(\vec{x}_n)_n$ ist kompatibel mit der Eingangsfolge $(\vec{u}_n)_n$, wenn $\vec{x}_{n+1} = F(\vec{x}_n, \vec{u}_{n+1}), \forall n \leq 0$ erfüllt ist.*

Definition 2 (Echo State Eigenschaft (ESP))**.** *Ein* ESN *$S : X \times U \to X$ besitzt die Echo State Eigenschaft genau dann wenn eine Nullfolge $(\delta_n)_{n \geq 0}$ existiert, sodass für alle Zustandsfolgen $(\vec{x}_n)_n, (\vec{x}'_n)_n$ die kompatibel mit der Eingangsfolge $(\vec{u}_n)_n$ sind gilt, dass $\forall n \geq 0 ||x_n - x'_n|| < \delta_n$*

Dies bedeutet, dass nachdem das Netzwerk lang genug betrieben worden ist, der Zustand nicht mehr von dem beliebig gewähltem Anfangszustand abhängt. Diese Eigenschaft ist notwendig, damit das ESN Vorhersagen treffen kann [3].

Nun stellt sich die Frage, wann ein Netzwerk diese Eigenschaft besitzt. Es wird schnell klar, dass dies nur durch die Gewichtsmatrix $\mathbf{W}$ bestimmt wird. Betrachtet man die Zustandsgleichung des Netzwerkes, so lässt sich auf Grund des *Banachschen Fixpunktsatzes* erkennen, dass die *ESP* für alle Eingänge $\vec{u}_n$ vorliegt, sobald $||\vec{x}_{n+1} - \vec{x}'_{n+1}|| < ||\vec{x}_n - \vec{x}'_n||$ für zwei kompatible Zustände $\vec{x}_n \neq \vec{x}'_n$ erfüllt ist [2]. Hieraus ergibt sich, dass die *ESP* vorliegt, wenn

$$|1 - \alpha(1 - \sigma_{max}(\mathbf{W}))| < 1 \tag{8}$$

erfüllt ist, wobei $\sigma_{max}(\mathbf{W})$ der größte Singulärwert ist [5].
Weitergehend ist bekannt, dass für Systeme bei denen der Spektralradius $\rho(\mathbf{W}) > 1$ ist diese Eigenschaft nicht vorliegen kann, sofern $\vec{u}_n = 0$ möglich ist [2, 5].

Hieraus ergab sich lange Zeit die falsche Annahme, dass für Systeme mit $\rho(\mathbf{W}) < 1$ die Eigenschaft stets garantiert ist. Wie allerdings gezeigt werden konnte, ist dies nicht der Fall [11]. Stattdessen konnte gezeigt werden, dass eine hinreichende Bedingung durch

$$\rho(\alpha|\mathbf{W}| + (1 - \alpha)\mathbf{I}) < 1 \tag{9}$$

gegeben ist - wobei als Betrag der Matrix hier das elementweise Betragsnehmen gemeint ist. Diese Bedingung ist weniger einschränkend als Gleichung (8) [11].

Darauf basierend kann nun eine Methode nach [11] angegeben werden, um die Gewichtsmatrix $\mathbf{W}$ zu konstruieren:

1. Generiere zufällige Matrix $\mathbf{W}$ mit $|\mathbf{W}| = \mathbf{W}$
2. Skaliere $\mathbf{W}$, sodass Gleichung (9) erfüllt ist.
3. Wechsel zufällig das Vorzeichen von ungefähr der Hälfte aller Einträge.

Statt dieser Vorschrift wurde zuvor oftmals $\mathbf{W}$ zufällig generiert und anschließend nur $\rho(\mathbf{W})$ statt $\rho(|\mathbf{W}|)$ skaliert, was mit unter zu instabilen Systemen geführt hat. Da allerdings auch für Systeme mit einem Spektralradius > 1 die *ESP* beobachtet werden kann für nicht verschwindende Eingänge $\vec{u}_n$, ist es ratsam auch effektive Spektralradien jenseits 1 auszuprobieren.

4 Anwendungen

Um den Aufwand für die spätere Benutzung der *ESN*s zu reduzieren, ist zuerst eine allgemeine Implementation dieser geschrieben worden. Diese wird nun anhand zweier exemplarischer Systeme getestet und die hierbei entstandenen Ergebnisse vorgestellt.

4.1 Vorhersage eines Rössler-Systems

Eine erste Anwendung, um die Möglichkeiten eines ESN zu demonstrieren, besteht in der Vorhersage des Verhaltens eines *Rössler-Systems*

$$\begin{aligned}\dot{x} &= -(y+z)\\ \dot{y} &= x + 0.25 \cdot y\\ \dot{z} &= 0.4 + (x - 8.5) \cdot z.\end{aligned} \tag{10}$$

Dieses System zeigt ein chaotisches Verhalten und stellt somit eine herausfordernde Aufgabe für die zeitliche Vorhersage dar.
Hierbei werden im Folgenden zwei verschiedene Aufgaben betrachtet: *(a)* die Vorhersage von $x'(t) \approx x(t+5)$ durch die Kenntnis von $x(t)$ und *(b)* die Kreuz-Vorhersage $y(t)$ durch $x(t)$. Für beide Aufgaben wurde das System für jeweils 6000 Schritte mit einer Abtastrate $\Delta t = 0.1$ in Analogie zu [9] simuliert, wobei die Hälfte der Daten für das Training und der Rest für die Testergebnisse verwendet worden ist.

4.1.1 Vorhersage $x(t) \rightarrow x(t+5)$

Für diese erste Aufgabe ist durch eine GRIDSEARCH ein geeignetes System gefunden werden, dessen Parameter in der Tabelle 2 dargestellt sind. Dafür wurde ein größerer Bereich des Hyperparameterraumes systematisch abgetastet und die jeweilige Leistung des sich ergebenen ESNs ermittelt, um anschließend die bestmöglichen Parameter zu wählen. Als Maß für diese Optimierung wurde der *Mean Square Error (MSE)*

$$MSE = \sum_n \left(x(t+5) - x'(t)\right)^2 \tag{11}$$

zwischen der Vorhersage $x'(t)$ und dem tatsächlichem Wert $x(t+5)$ benutzt.

Variable	Wert für *(a)*	Wert für *(b)*
N	300	500
α	0.25	0.20
$\rho(\lvert\mathbf{W}\rvert)$	0.80	3.00
ν_{max}	0.01	0.01
β	1×10^{-6}	–

Tab. 2: Auflistung der Parameter für das jeweilige ESN für (a) und (b).

Der Trainingsvorgang wurde mittels der *Tikhonov Regularisierung* nach Gleichung 6 durchgeführt. Im Training konnte ein Fehler von $MSE_{train} = 0.555$ erreicht werden, wobei sich im Testvorgang der Fehler zu $MSE_{test} = 0.237$ ergeben hat. Dieser ist im Vergleich zu dem Wertebereich $\approx [-15, 15]$ der Zeitreihe relativ klein - dies spricht für eine erfolgreiche Vorhersage. Ein graphischer Vergleich zwischen dem Signal $y(t)$ und der Vorhersage $v(n)$ ist in Abbildung 1 zu finden.

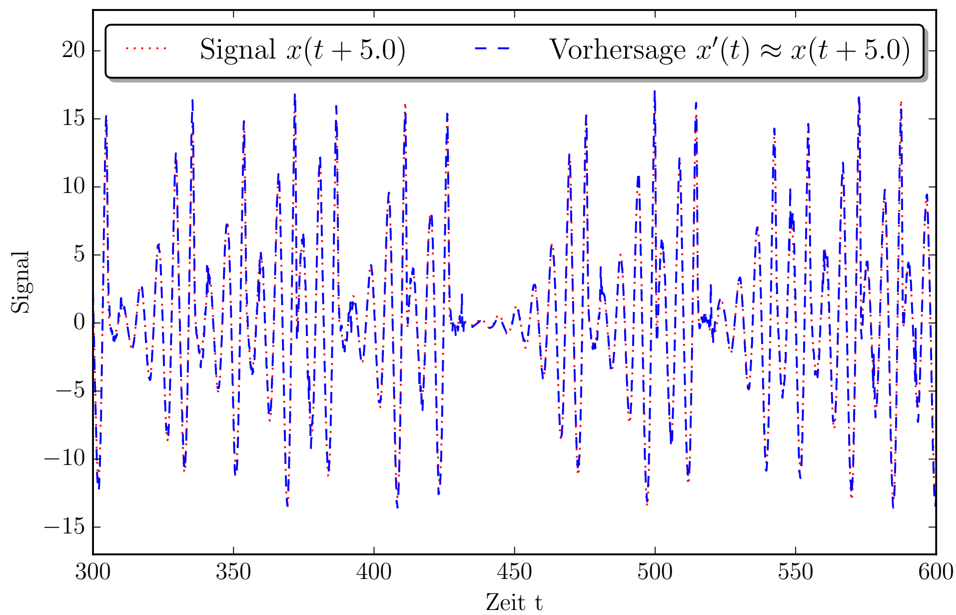

Abb. 1: Darstellung des Signals $x(t + 5.0)$ in rot und der Vorhersage $x'(t) \approx x(t + 5.0)$ in blau für das Rösler-System in der Testphase.

In dieser ist zu erkennen, dass die vorhergesagten Werte sehr nahe an den tatsächlichen Werten liegen und der makroskopische Verlauf der Zeitreihe gut wiedergegeben wird. Lediglich an einigen wenigen Stellen führt die vorhergesagte Zeitreihe $x'(t)$ eine Art Zitterbewegung durch, welche im richtigen Signal $x(t)$ nicht vorhanden ist.

4.1.2 Kreuz-Vorhersage $x(n) \rightarrow y(n)$

Auch für die Kreuz-Vorhersage von der $x(n)$ auf die $y(n)$ Zeitreihe konnte sehr schnell durch eine GRIDSEARCH ein geeignetes System gefunden werden. Als Maß wurde ebenfalls der *MSE* benutzt. Die gefundenen Parameter sind ebenso in der Tabelle 2 dargestellt. Interessant ist, dass der Spektralradius größer 1.0 ist. Dies widerspricht nicht direkt den Aussagen aus Abschnitt 3.4, da diese nur für Signale getroffen werden konnten, welche auch null sein können. Um dies auszuschließen ist ein konstanter *Bias* von 1.0 als weiteres Eingangssignal genutzt worden.
Für das Auslesen wurde eine Linearkombination genutzt, wobei die Lösung $\mathbf{W}_{out}$ mittels der Pseudoinversen nach Gleichung 7 bestimmt worden ist.

Mit dieser Methode wurde für den Trainingsvorgang ein Fehler $MSE_{train} = 6.2 \times 10^{-4}$ und für den Testvorgang von $MSE_{test} = 3.0 \times 10^{-5}$ bestimmt. Dies spricht wieder für eine gute Vorhersage. Die Darstellung der Vorhersage $y'(t)$ und des ermittelten Fehlers sind in den Abbildungen 2 und 3 zu finden.

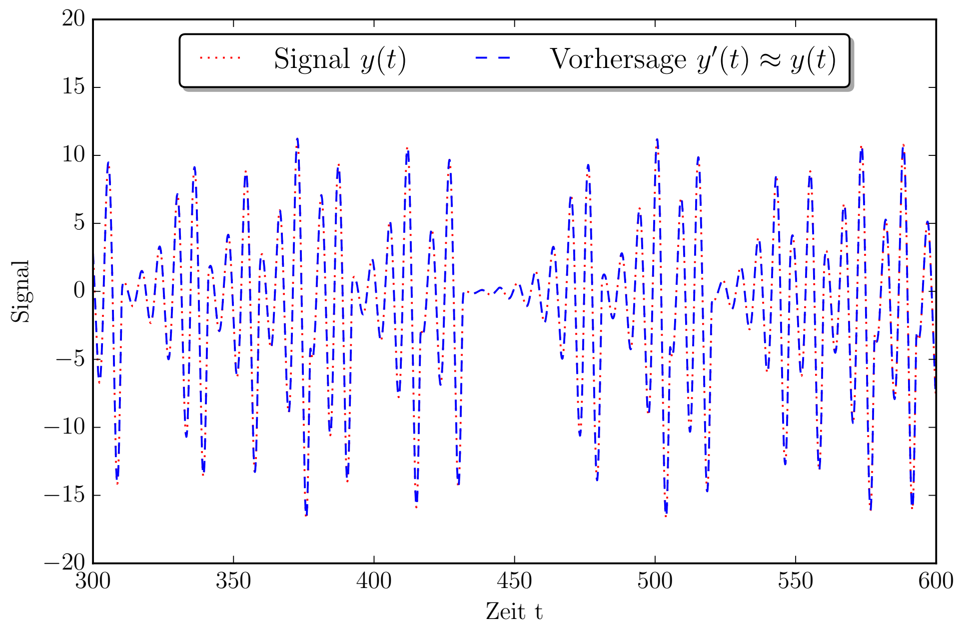

Abb. 2: Darstellung des Signals $y(t)$ in rot und der Vorhersage $y'(t)$ in blau für das Rössler-System in der Testphase.

In Abbildung 2 stimmt die Vorhersage $y'(t)$ augenscheinlich dem Signal $y(t)$ perfekt überein. Hierbei treten keine Zitterbewegungen mehr auf, die zuvor in Abschnitt 4.1.1 beobachtet worden sind.

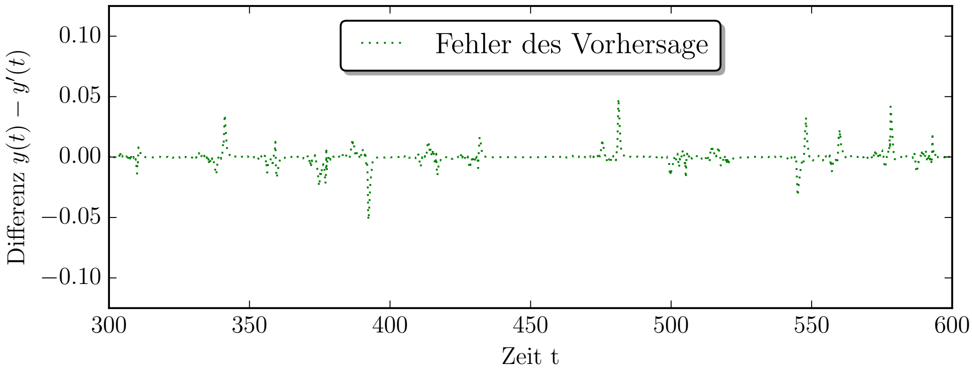

Abb. 3: Darstellung des Fehlers $y(t) - y'(t)$ in grün zwischen dem Signal $y(t)$ und der Vorhersage $y'(t)$ für das Rössler-System in der Testphase.

Die grafische Darstellung des Fehlers zeigt einen nahezu konstanten Verlauf mit nur wenigen Ausreißer, welche scharfe Peaks sind. Dies passt gut zu der qualitativen Analyse von Abbildung 3. Im Vergleich zu der vorhandenen Publikation [9] ergibt sich ein ähnlicher Verlauf des Fehlers. Leider können hierbei nur die Graphen verglichen werden, da keine expliziten Fehler angegeben sind.

4.2 Vorhersage eines Mackey-Glass-Systems

Als nächste Anwendung wird ein *Mackey-Glass-System*

$$\dot{x}(t) = \beta \frac{x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t) \tag{12}$$

betrachtet, wobei $\tau = 17, n = 10, \gamma = 0.1, \beta = 0.2$ gewählt worden sind, um die Ergebnisse abschließend mit den Resultaten von [7] vergleichen zu können. Dies ist eine beliebte Aufgabe im Bereich der Vorhersagen chaotischer Zeitreihen.
Um die Resultate mit [7] vergleichen zu können, wurde für das System in Analogie die Vorhersage $x(t) \rightarrow x(t+84)$ betrachtet. Dabei wurden die Parameter aus Tabelle 3 genutzt. Mittels der expliziten *Euler-Diskretisierung*

$$x(t+1) = x(t) + \Delta_t \cdot \left(\beta \frac{x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t)\right) \tag{13}$$

sind Werte für $0 \leq t \leq 4000$ simuliert worden, wobei $\Delta_t = 1 \times 10^{-2}$ die zeitliche Diskretisierungskonstante ist. Nun sind die ersten 2000 Werte für den Trainings und die anderen für den Test-Vorgang genutzt worden. Es wurde zur Lösung erneut die *Tikhonov Regularisierung* nach Gleichung 6 benutzt.

Variable	Wert
N	1000
α	0.20
$\rho(\mathbf{W})$	1.25
ν_{max}	1×10^{-4}
β	1×10^{-8}

Tab. 3: Auflistung der benutzten Parameter des ESN.

Auch hier ist wieder interessant, dass der Spektralradius größer 1.0 ist. Um dies sinnvoll nutzen zu können, ist ein konstanter *Bias* von 1.0 als weiteres Eingangssignal genutzt worden.
Der *MSE* wurde für die Trainingsphase als $MSE_{train} = 0.0042$ und für die Testphase als $MSE_{test} = 9 \times 10^{-5}$ bestimmt. Eine Darstellung der Ergebnisse ist in Abbildung 4 zu sehen.

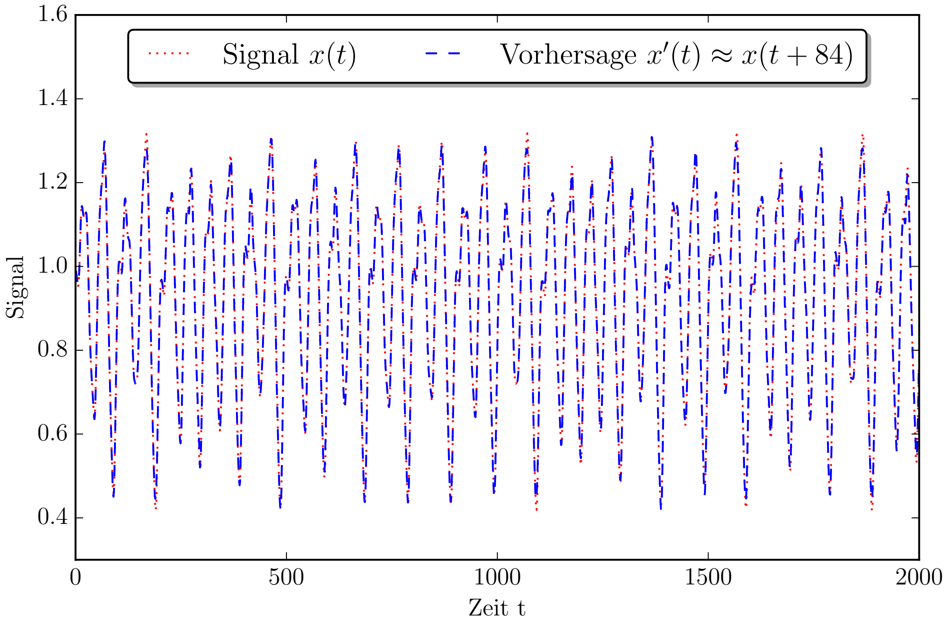

Abb. 4: Darstellung des Signals $x(t)$ in rot und der Vorhersage $x(t+84)$ in blau für das Mackey-Glass-System in der Testphase.

Es ist zu erkennen, dass die Vorhersage $x'(t)$ sehr nahe am tatsächlichen Signal $x(t)$ verläuft. Dies deckt sich mit dem sehr niedrigen *MSE* und spricht ebenfalls für eine gute Vorhersage. Dies ist eine starke Verbesserung im Vergleich zu dem besten Ergebnis aus [7] mit $MSE = 0.0014$, wobei anzumerken ist, dass dort eine kleinere Trainings- und Testphase gewählt worden ist.

5 Fazit

Wie die Ergebnisse aus Abschnitt 4 zeigen, bieten Echo State Networks eine gute Möglichkeit zur Vorhersage und Kreuz-Vorhersage von zeitlichen Messreihen. Hierbei konnten teilweise sogar in Abschnitt 4.2 die bisherigen Ergebnisse, welche mit anderen Methoden erzielt worden sind, übertroffen werden. Es hat sich gezeigt, dass für die meisten Anwendungen bereits eine kleine Reservoirgröße $N \leq 2000$ ausreichend ist. Dies hat zur Folge, dass sowohl der Trainings als auch der Testvorgang sehr schnell und ressourcenschonend durchgeführt werden konnten.

Ein Nachteil ist allerdings, dass die Modelle über viele Hyperparameter verfügen, welche alle anwendungsspezifisch angepasst werden müssen. In den meisten Fällen konnten diese allerdings relativ schnell so eingestellt werden, dass zufriedenstellende Ergebnisse erzielt werden konnten. Hierbei hat es sich als sehr hilfreich erwiesen erst grobe Parameterbereiche abzutasten, um eine Vorauswahl von vielversprechenden Hyperparametern zu finden, bevor diese fein abgetastet werden.
Für weitere Arbeiten könnte sich das Implementieren eines Algorithmus, der diese Parameter selbstständig schnell optimiert als hilfreich erweisen. Eine solche Methode ist in [5] vorgestellt worden, doch ist sie im Rahmen dieses Praktikums auf Grund der angegebenen Instabilität des Algorithmus nicht implementiert worden.

6 Ausblick

Für das Schreiben dieses Berichts hat sich die Wahl der LaTeX-Umgebung als gut geeignet herausgestellt. Die optische Darstellung von mathematischen Ausdrücken sowie des erklärenden Textes ist hierbei intuitiver umzusetzen als bei den bekannten Alternativen. Hierbei wurde die verwendete Literatur über Zotero und BibTeX direkt in den LaTeX-Quellcode eingebunden und zitiert.
Für die Literaturrecherche ist der Dienst Google Scholar benutzt worden, welcher eine umfassende Datenbank besitzt.

Das Programmieren der Anwendungsbeispiele wurde mit der Sprache Python und den bekannten Frameworks Numpy und Scipy umgesetzt. Diese Entscheidung wurde hauptsächlich durch die große Flexibilität dieser Sprache und der ausreichenden Leistung der Bibliotheken beeinflusst. Die hierbei entstandenen Grafiken wurden mit der Pyplot Bibliothek umgesetzt.

Dieses gesamte Vorgehen hat sich im Laufe des Praktikums als gut erwiesen und wird somit auch in der anschließenden Bachelorarbeit weiterverfolgt werden.

Literatur

[1] Cristopher M. Bishop. *Pattern Recognition andMachine Learning.* Springer, Cambridge, 2006. ISBN 0-387-31073-8.

[2] H. Jäger. The “echo state“ approach to analysing and training recurrent neural networks - with an erratum note. *GMD Report*, 148, 2001/2010.

[3] H. Jäger. A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the “echo state network“ approach. *GMD Report*, 159:48 ff., 2002.

[4] H. Jäger. Long short-term memory in echo state networks: Details of a simulation study. Technical report, Jacobs University Bremen - School of Engineering and Science, 2012.

[5] H. Jäger, M. Lukoševičiusa, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*, 20: 335–352, 2007.

[6] M. Lukoševičiusa and H. Jäger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[7] C. H. L´opez-Caraballo, I. Salfate, J. A. Lazz´us, P. Rojas, M Rivera, and L Palma-Chilla. Mackey–glass noisy chaotic time series prediction by a swarm-optimized neural network. *Journal of Physics: Conference Series*, 720(1), 2016.

[8] Wolfgang Maass. Liquid State Machines: Motivation, Theory, and Applications. In *Computability in Context*, pages 275–296. Imperial College Press, 2011.

[9] U. Parlitz and A. Hornstein. Dynamical prediction of chaotic time series. *Chaos and Complexity Letters*, 1(2):135–144, 2005.

[10] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, 28, 2013.

[11] I. Yildiz, H. Jäger, and S. Kiebel. Re-visiting the echo state property. *Neural Networks*, 35:1–9, 2012.