The Power of Linear Recurrent Neural Networks

Was können lineare rekurrente neuronale Netze?

Frieder Stolzenburg

▲ Hochschule Harz
Harz University of Applied Sciences

Hochschule Harz, Fachbereich Automatisierung und Informatik,
Friedrichstr. 57-59, 38855 Wernigerode, Deutschland

E-Mail: fstolzenburg@hs-harz.de

joint work with Oliver Obst, Olivia Michael, Sandra Litz, and Falk Schmidsberger
in the Decorating project (DEep COnceptors for tempoRal dATa mINinG)
funded by DAAD (Germany) and UA (Australia)

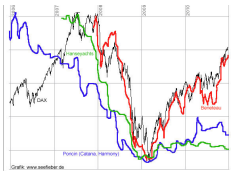
Overview

Time Series and Prediction

Definition

A time series is a series of data points in d dimensions $S(0), \ldots, S(n) \in \mathbb{R}^d$ where $d \geq 1$ and $n \geq 0$.

- **Examples:**
  - trajectories (of pedestrians, dance, sports, etc.)
  - stock quotations (of one or more companies)
  - weather forecast
  - natural language processing
    (speech recognition, text comprehension, question answering)
- **Time Series Analysis** allows
  - prediction of further values
  - data compression, i.e. compact representation
    (e.g. by a function $f(t)$)



Number Puzzles

- Number puzzles can be understood as one-dimensional time series.
- Such exercises often are part of
  - intelligence tests,
  - entrance examinations, or
  - job interviews.
- Examples: Which numbers continue the following series?
  1. 1,3,5,7,9,11,13,15 (arithmetic series)
  2. 1,2,4,8,16,32,64,128 (geometric series)
- Question:
  Can number puzzles be solved automatically by computer programs?
- We will do this by means of artificial recurrent neural networks (RNN), namely predictive neural networks with a reservoir of randomly connected neurons, which are related to echo state networks (ESNs) [1].

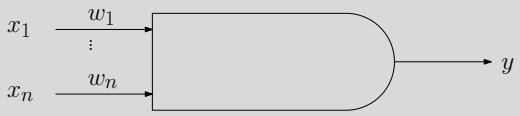
Artificial Neurons

- A recurrent neural network (RNN) is a directed graph (usually fully connected), i.e. an interconnected group of N nodes, called neurons.
- The activation of a neuron y at (discrete) time $t + \tau$ for some time step τ is computed from the activation of the neurons $x_1, \ldots, x_n$, that are connected to y with the weights $w_1, \ldots, w_n$, at time t:

$$y(t + \tau) \;=\; g\big(w_1 \cdot x_1(t) + \cdots + w_n \cdot x_n(t)\big)$$

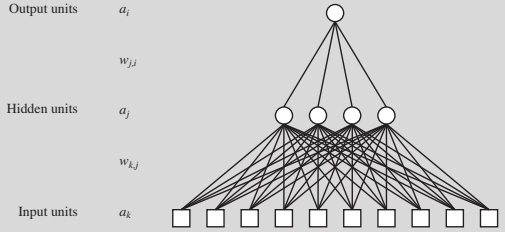
- g is called activation function.

Neural Unit

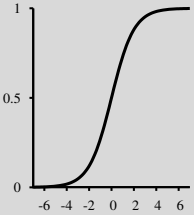


Feedforward Neural Nets

Multi-Layer Feedforward Network



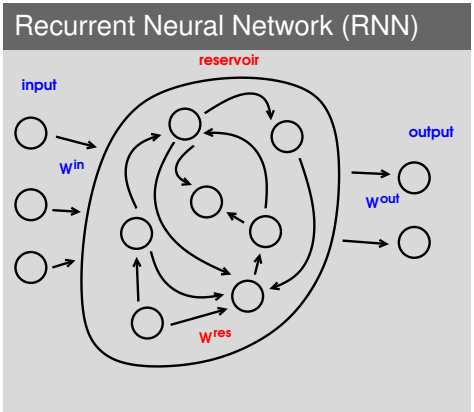
Sigmoidal Activation Function



- Network corresponds to directed acyclic graph with possibly multiple layers.
- Activation function g often is sigmoidal (i.e. non-linear threshold function).
- Complex functions can be learned by backpropagation (not here).
- There are no internal states in the network (no memory or time).

RNN Architecture

- In the reservoir, neurons may be connected recurrently.
- In the transition matrix W, an entry w_{ij} in row i and column j states the weight of the edge from neuron j to neuron i. If there is no connection, then $w_{ij} = 0$.
- Echo state networks [1]:
  - Input and reservoir weights W^{in} and W^{res} form random matrices (but with stationary dynamics [3]).
  - Only the weights leading to the output neurons W^{out} are learned.



Predictive Neural Networks

A predictive neural network (PrNN) is a RNN with the following properties:

1. For all neurons we have linear activation, i.e., everywhere g is the identity.
   - This simplifies learning a lot.
   - Still non-linear functions over time can be represented.
2. The weights in W^{in} and W^{res} are initially taken randomly, independently, and identically distributed from the standard normal distribution, whereas the output weights W^{out} are learned.
3. There is no clear distinction of input and output but only one joint group of d input/output neurons. They may be arbitrarily connected like the reservoir neurons.

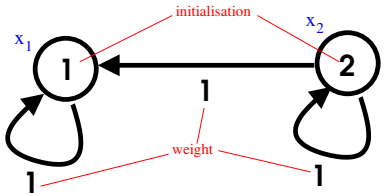
$$x_{\text{out}}(t) = x_{\text{in}}(t+1)$$

Example

- table :

t	0	1	2	3	4
$f(t)$	1	3	5	7	9

- function : $f(t) = ?2 \cdot t + 1$
- recursion :
  $f(0) = 1; \quad f(t+1) = f(t) + 2$
- network :



- matrix W and start vector x_0:

$$W = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad x_0 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- dynamics :

t	0	1	2	3	4
x_1	1	3	5	7	9
x_2	2	2	2	2	2

Remarks:

- $f(t) = W^t \cdot x_0$ (matrix product)
- spectral radius of $W^{\text{res}} \approx 1$
  (absolute value of largest eigenvalue)

Network Dynamics

Property 1

Let $W = V \cdot J \cdot V^{-1}$ be the Jordan decomposition of the transition matrix W (always existing) where J is the direct sum, i.e., a block diagonal matrix, of one or more Jordan blocks

$$J_m(\lambda) = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 \\ 0 & \lambda & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \lambda & 1 \\ 0 & \cdots & \cdots & 0 & \lambda \end{bmatrix}$$

in general with different sizes $m \times m$ and eigenvalues λ. Then it holds:

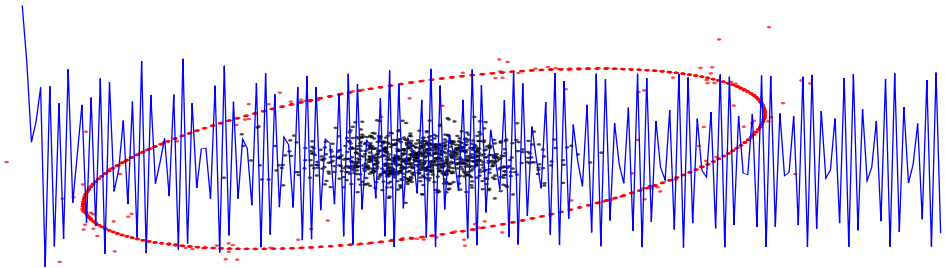
$$f(t) = W^t \cdot x_0 = V \cdot J^t \cdot V^{-1} \cdot x_0 \quad \left(= x_1 \, {\lambda_1}^t \, v_1 + \ldots + x_N \, {\lambda_N}^t \, v_N\right)$$

Long-Term Behaviour

Property 4

Let a recurrent neural network with random, real-valued transition matrix W and spectral radius 1, i.e. $|\lambda^{\max}| = 1$, be given (e.g. a pure reservoir). In the long run, the network states $f(t)$ either

1. move into a singularity ($\lambda^{\max} = +1$),
2. oscillate between two points ($\lambda^{\max} = -1$), or
3. rotate in two dimensions on an ellipse with a uniform angular frequency ($\lambda^{\max}_{1,2} \in \mathbb{C}$). [7]



Network Learning

- We want to predict the next value(s) of time series.
- For predictive neural networks, we have:
  $x_{\text{out}}(t) = x_{\text{in}}(t+1)$ (i.e. input = output)
- We take a reservoir with N random neurons (here $N = 1$, simplified for didactic reasons).
- At each time point t it holds (assuming linear dependency):

$$x_{\text{out}}(t) = W^{\text{out}} \cdot \begin{pmatrix} x_{\text{in}}(t) \\ x_{\text{res}}(t) \end{pmatrix} \text{ with } W^{\text{out}} = (w_{\text{in}}\ w_{\text{res}})$$

- Thus, together with the input, the reservoir is used for auxiliary computations.
- We just have to solve a linear equation system.

Running Example

t	0	1	2	3	4
x_{in}	1	3	5	7	9
x_{res}	2	2	2	2	2
x_{out}	3	5	7	9	?

Learning and Representation

Property 6

From a real-valued function $f(t)$, possibly in multiple dimensions, let a series of function values $f(t_0), \ldots, f(t_n)$ be given. Then there is a predictive neural network with the following properties:

1. It runs exactly through all given $n+1$ function values, i.e. it approximates $f(t)$.
2. It can effectively be learned.

We have to take enough reservoir neurons: $N^{\text{res}} \geq n - d$ (with d number of input dimensions).

- Property is related to the universal approximation theorem for non-recurrent neural networks: one linear output layer and one hidden layer activated by a non-linear function is needed
- For PrNNs, linearly activated units suffice without exception, but the approximated function has only one-dimensional input, namely t.
- Function $f(t)$ may be learned effectively. No iterative method like backpropagation is required, we just have to solve a linear equation system.

Dimension Reduction

- The number N^{res} of required reservoir neurons may be very high.
- Question: Can the resulting transition matrix $W = \begin{bmatrix} & W^{\text{out}} \\ W^{\text{in}} & W^{\text{res}} \end{bmatrix}$ be reduced?
- Idea: Reduce dimensionality of transition matrix W afterwards.
- For ESNs, there is a similar idea, namely conceptors [2], which however reduce only the spatial dimensionality of the point cloud.
- We reduce transition matrix W, respecting temporal order of data points.

Property 7

- The transition matrix W can be transformed by repeatedly applying Prop. 1 (and 5).
- The Jordan matrix can be used as sparse transition matrix.
- Non-relevant Jordan components can be deleted, as long as error < given threshold.

Algorithm

% d-dimensional function, given sampled, as time series
$\mathrm{S} = [f(0) \ldots f(n)]$

% random initialization of reservoir and input weights
$W^{\text{in}} = \text{randn}(N, d)$
$W^{\text{res}} = \text{randn}(N^{\text{res}}, N^{\text{res}})$

% learn output weights by linear regression
$X = \left[W^t \cdot s \right]_{t=0,\ldots,n}$
$Y^{\text{out}} = \left[S(1) \cdots S(n) \right]$
$W^{\text{out}} = Y^{\text{out}} / X$

% transition matrix and its decomposition
$W = \begin{bmatrix} & W^{\text{out}} \\ W^{\text{in}} & W^{\text{res}} \end{bmatrix}$
$J = \text{jordan\_matrix}(W)$

% network size reduction
$y = \left[1 \cdots 1 \right]^\dagger$
$Y = \left[J^t \cdot y \right]_{t=0,\ldots,n}$
$A = X / Y$ with rows restricted to input/output dimensions
reduce(A, J, y) to relevant components
such that RMSE$(S,\text{Out}) < \theta$

Dimension Reduction (continued)

Property 8

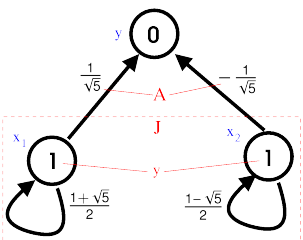
The time complexity is just $O(N^3)$ for both output weights learning and (one step of) dimensionality reduction. In practice, the complexity depends on the bit length of numbers in floating point arithmetics, and may be worse hence. The size of the learned network is in $O(N)$.

Number Puzzles (revisited)

- 1,3,5,7,9,11,13,15 (arithmetic series)
- 1,2,4,8,16,32,64,128 (geometric series)
- 1,3,6,10,15,21,28,36 (triangular numbers)
- 1,1,2,3,5,8,13,21 (Fibonacci series)

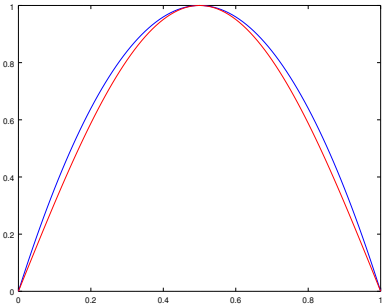
We obtain small, efficient, and sparsely connected networks,

e.g. Fibonacci: $\rightsquigarrow$ Moivre-Binet formula: $\frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^t - \left(\frac{1-\sqrt{5}}{2} \right)^t \right)$



Example

- Exercise: How can the curves be continued?
- What functions are shown?
  blue: $f(t) = 4\,t(1-t)$ (parabola)
  red: $f(t) = \sin(\pi\,t)$ (sine)
- Procedure:
  1. Sample curve (here) for $t = [0;1]$ with $\tau = 0.01$.
  2. Learn output weights W^{out} starting with a large enough reservoir, i.e. N^{res} big.
  3. Reduce number N of dimensions of transition matrix W (obtaining $\hat{W}^{\text{out}}$, $\hat{D}$, and $\hat{x}$).



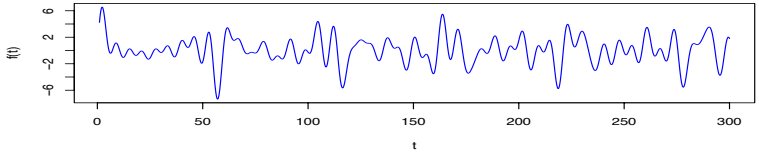
- Both functions can be learned and are discriminated correctly.
- For polynomials, eigenvalues are clustered in Jordan blocks (parabola, $N = 3$).
- For ellipses (sinusoids) $f(t) = \begin{pmatrix} a\,\cos(\rho\,t) \\ b\,\sin(\rho\,t) \end{pmatrix}$, we need only $N = 2$ neurons:

$$f(0) = \begin{pmatrix} a \\ 0 \end{pmatrix} \text{ and } f(t+\tau) = \begin{pmatrix} \cos(\rho) & -a/b\,\sin(\rho) \\ b/a\,\sin(\rho) & \cos(\rho) \end{pmatrix} \cdot f(t) \text{ where } \rho = \text{ angle of rotation}$$

Multiple Superimposed Oscillators

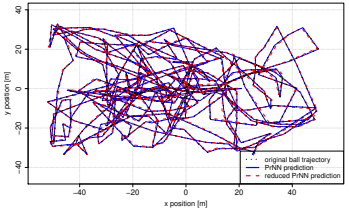
- MSO count as difficult benchmark problems for RNNs [3].
- **Definition:** $S(t) = \sum_{k=1}^{n} \sin(\alpha_k\, t)$

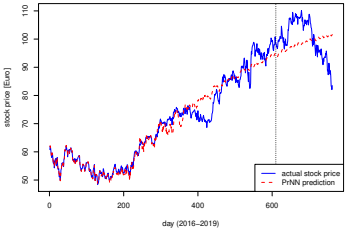
  MSO8: $n = 8$ and $\alpha_k \in \{0.2, 0.311, 0.42, 0.51, 0.63, 0.74, 0.85, 0.97\}$
- PrNN learning procedure arrives at $N = 16$ neurons – minimal size.
- Thus PrNNs outperform the previous state-of-the-art for the MSO task with minimal number of units. [3] report $N = 68$ as optimal reservoir size for ESNs.
- PrNN with $2n$ neurons suffices to represent a signal of n sinusoids [6].



Summary

- Predicting time series can be done by PrNNs.
- Only a linear equation system has to be solved.
- No backpropagation or similar procedure is required.
- Dimension reduction is possible efficiently.
- Matlab/Octave implementation exists
  Python – work in progress.
- Applications:
  1. mobile robotics (RoboCup simulation league) [4]
     average size reduction 29.2% for RMSE < 1 m
  2. predicting stock prices
     average deviation 6.1% (test set = 1/5 of year)





RNNs for Cognitive Reasoning

- Cognitive Reasoning addresses Commonsense Reasoning problems by neural networks + knowledge representation.
- Benchmarks are used:

Copa (Choices of Plausible Alternatives)

Q: My body cast a shadow over the grass. What was the cause?
A1: The sun was rising.
A2: The grass was cut.

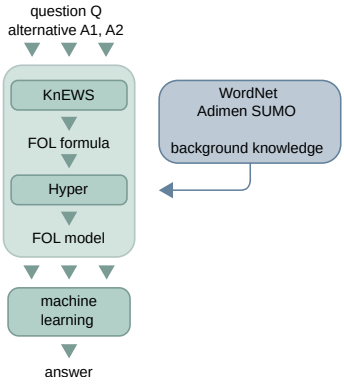
- KnEWS (Discourse Representation Theory):

```
fol(1,some(A,and(sun(A),some(B,and(r1Actor(B,A),rise(B))))))).
```

- Hyper (partial first-order logic model)

```
sun(sk1).
r1Actor(sk2,sk1).
rise(sk2).

p_d_disjoint(c_AstronomicalBody, c_Motion).
```



joint work with Sophie Siebert, Claudia Schon, and Ulrich Furbach in the CoRg project (Cognitive Reasoning) funded by DFG

RNNs for Cognitive Reasoning (to be continued) [5]

Future Work

- generalisation to other tasks than prediction: classification, behavior recognition, reinforcement learning
- combine knowledge representation and machine learning by neural-symbolic reasoning approach for XAI (explainable AI)
- **Thank You!**

References

[1] H. Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007. `http://www.scholarpedia.org/article/Echo_state_network`.

[2] H. Jaeger. Controlling recurrent neural networks by conceptors. CoRR – Computing Research Repository `http://arxiv.org/abs/1403.3369`, Cornell University Library, 2014.

[3] D. Koryakin, J. Lohmann, and M. V. Butz. Balanced echo state networks. *Neural Networks*, 36:35–45, 2012.

[4] O. Michael, O. Obst, F. Schmidsberger, and F. Stolzenburg. Analysing soccer games with clustering and conceptors. In H. Akyama, O. Obst, C. Sammut, and F. Tonidandel, editors, *RoboCup 2017: Robot Soccer World Cup XXI. RoboCup International Symposium*, LNAI 11175, pages 120–131, Nagoya, Japan, 2018. Springer Nature Switzerland.

[5] S. Siebert, C. Schon, and F. Stolzenburg. Commonsense reasoning using theorem proving and machine learning. In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, editors, *Machine Learning and Knowledge Extraction – 3rd IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2019*, LNCS 11713, pages 395–413, Canterbury, UK, 2019. Springer Nature Switzerland.

[6] F. Stolzenburg. Periodicity detection by neural transformation. In E. Van Dyck, editor, *ESCOM 2017 – 25th Anniversary Conference of the European Society for the Cognitive Sciences of Music*, pages 159–162, Ghent, Belgium, 2017. IPEM, Ghent University. Proceedings.

[7] F. Stolzenburg, S. Litz, O. Michael, and O. Obst. The power of linear recurrent neural networks. In D. Brunner, H. Jaeger, S. Parkin, and G. Pipa, editors, *Cognitive Computing – Merging Concepts with Hardware*, Hannover, 2018. Received *Prize for Most Technologically Feasible Poster Contribution*. Latest revision 2020.