A View of Reservoirs as State Space Models

Peter Tiňo

School of Computer Science
University of Birmingham, UK
Special thanks to ...

- Barbara Hammer
- Alessio Micheli
- Michal Čerňanský
- Luba Beňúšková
- Jort van Mourik
- Igor Farkaš
- Ali Rodan
- Jochen Steil
- Ata Kabán
- Nick Gianniotis
- Dan Conford
- Manfred Opper
- ...

...
Making sense of time series

"the environment"

...abbacbaaaccbdaabdccccca...

hmmm, how can I make some sense out of this...

What is your task?

modelling prediction classification...

A View of Reservoirs as State Space Models – p.3/65
Let’s start simple. We have

- a finite alphabet of abstract symbols $A = \{1, 2, ..., A\}$
- possibly infinite strings over $A$
- left-to-right processing of strings (as opposed to parsing the strings)
- a non-autonomous dynamical system in which the input stream driven trajectories evolve - “computation via state-space trajectories”
Left-to-right processing of sequences

One can attempt to reduce complexity of the processing task

group together histories of symbols that have “the same functionality” w.r.t. the given task (e.g. next-symbol prediction)

Information processing states (IPS) are equivalence classes over sequences

IPS code what is important in everything we have seen in the past
"Grammatical"/class 1: odd number of b’s
"Non-Grammatical"/class 0: even number of b’s (including none)
translate input streams over \( \{a, b\} \) to sequences over \( \{x, y\} \)

![Diagram of transducer]

1

2

\( a | x \)

\( b | y \)

\( b | x \)

\( a | x \)
The simplest construction of IPS is based on concentrating on the very recent (finite) past.

Example:
Only the last 5 input symbols matter when reasoning about what symbol comes next.

\[
\begin{align*}
\ldots & 1 2 1 1 2 3 2 1 2 1 2 4 3 2 1 1 \\
\ldots & 2 1 1 1 1 3 4 4 4 1 4 3 2 1 1 \\
\ldots & 3 3 2 2 1 2 1 2 2 1 3 4 3 2 1 1
\end{align*}
\]

All three sequences belong to the same IPS “43211”
finite context-conditional next-symbol distributions

\[ P(s \mid 11111) \]
\[ P(s \mid 11112) \]
\[ P(s \mid 11113) \]
\[ \ldots \]
\[ P(s \mid 11121) \]
\[ \ldots \]
\[ P(s \mid 43211) \]
\[ \ldots \]
\[ P(s \mid 44444) \]

\[ s \in \{1, 2, 3, 4\} \]
Difficult times...

but, who is going to pay for all this?

difficult times, pay for necessary things only!
Markov model

MM are intuitive and simple, but ... 

Number of IPS (prediction contexts) grows exponentially fast with memory length

Large alphabets and long memories are infeasible:
- computationally demanding and
- very long sequences are needed to obtain sound statistical estimates of free parameters
Variable memory length MM (VLMM)

Sophisticated implementation of potentially high-order MM

Takes advantage of subsequence structure in data

Saves resources by making memory depth context dependent

Use deep memory only when it is needed

Natural representation of IPS in form of Prediction Suffix Trees

Closely linked to the idea of universal simulation of information sources.
Prediction suffix tree (PST)

IFS are organized on nodes of PST
Traverse the tree from root towards leaves in reversed order
Complex and potentially time-consuming training

Each node $i$
has associated next-symbol probability distribution
$P(. | i)$
Continuous state space

States of the DS form IPS

These states code the entire history of symbols we have seen so far
The power of contractive DS

A View of Reservoirs as State Space Models – p.15/65
Affine input-driven dynamics

\[ x_t = f(u_t, x_{t-1}) = \rho x_{t-1} + (1 - \rho)u_t. \]

or, more generally

\[ x_t = \frac{\rho}{L} x_{t-1} + u_t, \quad u_t \in \left\{ 0, \frac{1}{L}, \frac{2}{L}, \ldots, \frac{L-1}{L} \right\}^d, \]

with a scale parameter \( \rho \in (0, 1] \).
Affine input-driven dynamics - cont’d

\[
\begin{array}{ccc}
7 & 8 & 9 \\
4 & 5 & 6 \\
1 & 2 & 3 \\
n=1
\end{array}
\quad\quad
\begin{array}{ccc}
77 & 87 & 97 \\
47 & 57 & 67 \\
17 & 27 & 37 \\
n=2
\end{array}
\]
Mapping sequences with contractions

Mapping symbolic sequences into Euclidean space

Coding strategy is Markovian

Sequences with shared histories of recently observed symbols have close images

The longer is the common suffix of two sequences, the closer are they mapped to each other
Cluster structure of IPS

Because IPS leading to the same/similar output lie close to each other, it is natural to look at the cluster structure of IPS.

Each cluster represents an abstract IPS.

If we wish, we can estimate the topology/statistics of state transitions/symbol emissions on such new IPS on the available data and test them on a hold-out set.
Each cluster $i=1,2,...,M$ has an associated next-symbol probability distribution $P(\cdot|i)$.
Fractal Prediction Machine

Each cluster $i=1,2,\ldots,M$ has associated next-symbol probability distribution $P(.|i)$. 

Cluster IPS

Delay

A View of Reservoirs as State Space Models – p.21/65
Contractive affine input driven dynamics is naturally related to VLMM

Memory resources are used efficiently:

- many quantization centers cover dense clusters – sequences with long suffixes
- very few centers in sparse areas – don’t waste memory on subsequences barely present in the data
Training sequence: 8000 symbols
Test sequence: 2000 symbols
Fractal representation through affine dynamics

CBR of Laser data

FPM contexts

PST contexts

MM contexts
Laser - results (NNL on test set)

NNL of FPMs and MMss on Laser data

NNL of PSTs on Laser data
NNL of MMs, FPMs and PST on a test text from Genesis

Training sequence: Old Testament - Genesis
Test sequence: Genesis
Things can be much more complicated
May need to remember events that happened long time ago
*Time lag* into the past can be *unbounded*
In such a state-transition formulation, finite number of IPS may not be enough.

E.g. when moving from regular languages to context-free languages, if we want to keep finite state formulation, we need an additional memory with LIFO organization.

Example:
Match the counts
$1^n2^n, \ n \geq 1$

More complicated: context sensitive, Turing computable.
There are discrete finite state models for dealing with sequential structures of increasing (Chomsky) complexity.

Apart from the basic finite state transition skeleton they contain additional memory mechanism of increasing complexity.

Algorithms for learning such representations (deterministic/probabilistic) from training data, although in general, as with all flexible models fitted on limited data, this is quite problematic (e.g. Gold)
Another possibility is to keep the state transition formulation without additional memory mechanism, but allow infinite number of IFS (countable, or even uncountable).

E.g. Dynamical recognizers studied by Pollack, Moore, ...

State transitions:
On symbol 1: \( x \rightarrow F_1(x) \)
On symbol 2: \( x \rightarrow F_2(x) \)

12122 – grammatical
221 – non-grammatical
Continuous state space

State transitions over a continuous state space $\mathcal{X}$ can be considered iterations of a non-autonomous dynamical system on $\mathcal{X}$.

Each map $F_i : \mathcal{X} \rightarrow \mathcal{X}$ corresponds to distinct input symbol $i \in \mathcal{A}$.

Having a state space with uncountable number of states can lead to Turing (super-Turing) capabilities, but we need infinite precision of computations (Siegelmann, Moore,...).
(Simple) Recurrent Neural Network

Neurons organized in 4 groups: Input (U), Context (C), Recurrent (X) and Output (Y)

X+C can be considered an “extended” input to the network

C is an exact copy of X from the previous time step
\[ x(t) = f(u(t), c(t)) = f(u(t), x(t - 1)) \]

\[ y(t) = h(x(t)) \]

\[ x_i(t) = \sigma \left( \sum_j W_{ij}^{RI} \cdot u_j(t) + \sum_j W_{ij}^{RC} \cdot c_j(t) + T_i^R \right) \]

\[ y_i(t) = \sigma \left( \sum_j W_{ij}^{OR} \cdot x_j(t) + T_i^O \right) \]

\( W^{RI}, W^{RC}, W^{OR} \) and \( T^R, T^O \) are real-valued connection weight matrices and threshold vectors, respectively
\( \sigma \) is a sigmoid activation function, e.g. \( \sigma(u) = (1 + \exp(-u))^{-1} \)
Attractive sets

To latch a piece of information for a potentially unbounded number of time steps we need attractive sets.

Grammatical:
all strings containing odd number of 2’s
1. To latch an important piece of information for future reference we need to create an attractive set.

2. But derivatives of the state-transition map are small in the neighborhood of such an attractive set.

3. We cannot propagate error information when training RNN via a gradient-based procedure – derivatives decrease exponentially fast through time.
Curse of long time spans

As soon as we try to create a mechanism for keeping important bits of information from the past, we lose the ability to set the parameters to the appropriate values since the error information gets lost very fast.

It is actually quite non-trivial (although not impossible) to train RNN beyond finite memory on non-toy problems.
The State → Output map is often quite smooth and sometimes monotonic, e.g. squashed version of a linear transformation.

IPS leading to the same/similar output are forced to lie close to each other in the state space.
Map $y(t) = h(x(t))$ is a squashed version of a linear transformation, and hence it is quite smooth and monotonic.

IPS $x(t)$ leading to the same/similar output are forced to lie close to each other in the RNN state space.
Frequent and very popular heuristics for enhancing RNN generalization are based on clustering RNN state space into a finite number of clusters.

Each cluster represents an abstract IPS.

Topology/statistics of state transitions/symbol emissions on such new IPS are estimated on the available data and tested on a hold-out set.
Clustering IPS

Extracted FSM:
all strings containing odd number of 2’s
Neural Prediction Machines (NPM)

Each cluster $i=1,2,...,M$ has associated next-symbol probability distribution $P( . | i )$.

Cluster $X$

Recurrent $X$

Input $U$

Context $C$

... 1 2 2 3 2 1 1 1 2
RNN without any training is already great!

There are good reasons for initializing RNN with small weights (indeed it is a common practice)

State-transition maps of RNN (with commonly used sigmoid activation functions) prior to training (small weights) are contractions

Clustering IPS naturally forms Markov states: we group together neighboring points that are images of sequences with shared suffixes
Given an input symbol $s \in \mathcal{A}$ at time $t$, and activations of recurrent units from the previous step, $\mathbf{x}(t-1)$,

$$\mathbf{x}(t) = f_s(\mathbf{x}(t-1)).$$

This notation can be extended to sequences over $\mathcal{A}$ the recurrent activations after $t$ time steps are

$$\mathbf{x}(t) = f_{s_1...s_t}(\mathbf{x}(0)).$$

If maps $f_s$ are contractions with Lip. constant $0 < C < 1$,

$$\|f_{vq}(\mathbf{x}) - f_{wq}(\mathbf{x})\| \leq C^{\|q\|} \cdot \|f_v(\mathbf{x}) - f_w(\mathbf{x})\|$$

$$\leq C^{\|q\|} \cdot diam(\mathcal{X}).$$
Chaotic laser revisited

(a) Markov models on laser

(b) RNN output on laser

(c) RTRL - NPM on laser - 16 recurrent neurons

(d) EKF - NPM on laser - 16 recurrent neurons
Deep Recursion

Production rules:
\[ R \rightarrow 1R3, \quad R \rightarrow 2R4, \quad R \rightarrow e. \]

<table>
<thead>
<tr>
<th>Length</th>
<th>Percent in the sequence</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>34.5 %</td>
<td>13, 24</td>
</tr>
<tr>
<td>4</td>
<td>20.6%</td>
<td>1243, 1133</td>
</tr>
<tr>
<td>6</td>
<td>13.6 %</td>
<td>122443, 211344</td>
</tr>
<tr>
<td>8</td>
<td>10.3%</td>
<td>22213444</td>
</tr>
<tr>
<td>10,12,14,...20</td>
<td>each length 3.5 %</td>
<td>12121211133343434343</td>
</tr>
</tbody>
</table>

Table 1: Distribution of string lengths. Training sequence – 6156 symbols, test sequence – 6192 symbols
Deep recursion - results

(a) Markov models on context-free language

(b) RNN output on context-free language

(c) RTRL - NPM on CFL - 16 recurrent neurons

(d) EKF - NPM on CFL - 16 recurrent neurons

A View of Reservoirs as State Space Models – p.46/65
Theoretical grounding

Theorem (Hammer & Tiňo):

- Recurrent networks with contractive transition function can be approximated arbitrarily well on input sequences of unbounded length by a definite memory machine. Conversely,

- every definite memory machine can be simulated by a recurrent network with contractive transition function.

Hence initialization with small weights induces an architectural bias into learning with recurrent neural networks.
Learnability (PAC framework)

The architectural bias emphasizes one possible region of the weight space where generalization ability can be formally proved.

Standard RNN are not distribution independent learnable in the PAC sense if arbitrary precision and inputs are considered.

Theorem (Hammer & Tiño):

- Recurrent networks with contractive transition function with a fixed contraction parameter fulfill the distribution independent UCED property and so

- unlike general recurrent networks, are distribution independent PAC-learnable.
Fractal analysis

Complexity of the driving input stream (topological entropy) is directly reflected in the complexity of the state space activations (fractal dimension).

Theorem (Tiňo & Hammer):
Recurrent activations inside contractive RNN form fractal clusters the dimension of which can be bounded by the scaled entropy of the underlying driving source. The scaling factors are fixed and are given by the RNN parameters.
So let’s not bother to train the dynamical part!

Fix the DS part to a contraction.

Only train the simple linear readout output map. Can be done efficiently.

Reservoir computation models.
Fix the DS part to:

- **standard RNN dynamics** - Echo State Network (ESN)
  - randomized interconnection structure - random interconnection weights

- **recurrent spiking network** - Liquid State Machine (LSM)
  - readout operates on time integrated activations

- **affine contractions** - Fractal Prediction Machine (FPM)
  - readout is a collection of spatially distributed multinomials

- real bucket of water, bacteria metabolic pathway ...
ESN/LSM - This is great and it ‘works’, BUT ...

- Having to rely on random reservoir construction makes me nervous.
  - Exactly why is it that we need random elements in reservoir construction?
  - If we need randomness, how can its use be minimized so that “good reservoirs” are still obtained?

- Deterministically constructed simple reservoirs are ameanable to theoretical analysis.
  - How is the input stream translated into reservoir activations?
  - When are the reservoir representations of historical contexts in the input stream useful and when not?
Still playing the devil’s advocate...

- Where does the “universality” of reservoirs come from?
  - To what extend is “diversity of computational elements in reservoir” necessary?
  - What features of reservoirs are important for their “universality”?

- How can we usefully characterize the reservoirs?
  - Memory quantification: Short term memory capacity, Fisher memory.
  - Properties of the reservoir connection matrix: Spectral radius, sparsity, eigenspectrum.
  - Input stream dependent measures: “edge-of-chaos” computations, kernel separation.
Simple, non-randomized reservoirs

Dynamical Reservoir
N internal units
\[ x(t) \]
Input unit \( s(t) \)
Output unit \( y(t) \)

\[ V \]
\[ U \]
\[ W \]
Simple reservoir models - some results

A View of Reservoirs as State Space Models – p.55/65
Simple reservoir models - some results

A View of Reservoirs as State Space Models – p.56/65
Simple reservoir models

deterministic input weight structure

A View of Reservoirs as State Space Models – p.57/65
Memory capacity

[Jaeger] A measure correlating past events in an i.i.d. input stream with the network output.

Assume the network is driven by a univariate stationary input signal $u(t)$.

For a given delay $k > 0$, construct the network with optimal parameters for the task of outputting $u(t - k)$ after seeing the input stream $...u(t - 1)u(t)$ up to time $t$.

Goodness of fit - squared correlation coefficient between the desired output $u(t - k)$ and the observed network output $y(t)$.
Memory capacity

\[ MC_k = \frac{\text{Cov}^2(u(t - k), y(t))}{\text{Var}(u(t)) \text{ Var}(y(t))}, \]

where \( \text{Cov} \) denotes the covariance and \( \text{Var} \) the variance operators. The short term memory (STM) capacity is then given by:

\[ MC = \sum_{k=1}^{\infty} MC_k. \]

[Jaeger] For any recurrent neural network with \( N \) recurrent neurons, under the assumption of i.i.d. input stream, the STM capacity cannot exceed \( N \).
Memory capacity of SCR

[Tiño & Rodan] Under the assumption of zero-mean i.i.d. input stream, the memory capacity of linear SCR architecture with $N$ reservoir units can be made arbitrarily close to $N$.

Theorem (Tiño & Rodan): Consider a linear SCR network with reservoir weight $0 < r < 1$ and an input weight vector such that the reservoir activations are full rank. Then, the SCR network memory capacity is equal to

$$MC = N - (1 - r^{2N}).$$
Cycle reservoir with regular jumps

still a simple deterministic construction...

(A) N=18

1

(BC) N=18

1

10

13

16

r_c

r_j

r_j

r_c

r_c

r_c

r_j

r_j

r_j

r_c

r_c

r_c

r_c

r_c

r_c

r_c

r_c

r_c
### CRJ - NARMA(10), laser, speech recognition

<table>
<thead>
<tr>
<th>reservoir model</th>
<th>$N = 100$</th>
<th>$N = 200$</th>
<th>$N = 300$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESN</td>
<td>0.0788 (0.00937)</td>
<td>0.0531 (0.00198)</td>
<td>0.0246 (0.00142)</td>
</tr>
<tr>
<td>SCR</td>
<td>0.0868</td>
<td>0.0621</td>
<td>0.0383</td>
</tr>
<tr>
<td>CRJ</td>
<td><strong>0.0619</strong></td>
<td><strong>0.0196</strong></td>
<td><strong>0.0130</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reservoir model</th>
<th>$N = 100$</th>
<th>$N = 200$</th>
<th>$N = 300$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESN</td>
<td>0.0128 (0.00371)</td>
<td>0.0108 (0.00149)</td>
<td>0.0089 (0.0017)</td>
</tr>
<tr>
<td>SCR</td>
<td>0.0139</td>
<td>0.0112</td>
<td>0.0106</td>
</tr>
<tr>
<td>CRJ</td>
<td><strong>0.00921</strong></td>
<td><strong>0.00673</strong></td>
<td><strong>0.00662</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reservoir model</th>
<th>$N = 100$</th>
<th>$N = 200$</th>
<th>$N = 300$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESN</td>
<td>0.0296 (0.0063)</td>
<td>0.0138 (0.0042)</td>
<td>0.0092 (0.0037)</td>
</tr>
<tr>
<td>SCR</td>
<td>0.0329 (0.0031)</td>
<td>0.0156 (0.0035)</td>
<td>0.0081 (0.0022)</td>
</tr>
<tr>
<td>CRJ</td>
<td><strong>0.0281 (0.0032)</strong></td>
<td><strong>0.0117 (0.0029)</strong></td>
<td><strong>0.0046 (0.0021)</strong></td>
</tr>
</tbody>
</table>
Reservoir characterizations

Eigen-decomposition of reservoir matrix

![Eigen-decomposition plots for different reservoirs](image-url)
Reservoir characterizations

Memory capacity

(A) (B) (C) (D)
Reservoir characterizations

Pseudo-Lyapunov exponents

![Graphs showing Pseudo-Lyapunov exponents for different tasks and scaling parameters.](image-url)