Markovian Architectural Bias of Recurrent Neural Networks

Peter Tiño

www.cs.bham.ac.uk/~pxt

School of Computer Science
University of Birmingham

2003
a collaboration with ...

- Barbara Hammer
- Michal Čerňanský
- Luba Buňúšková
Task

- Given a finite alphabet of symbols $A = \{1, 2, \ldots, A\}$
- Assume some sort of structure over sequences over $A$, e.g. language, generating source, etc.
- Learn about the underlying structure from example sequences
Information processing states

Process sequences by possibly grouping together histories of symbols that have “the same functionality” w.r.t. the given task.

Information processing states (IPS) are identifiers of the groups (equivalence classes over sequences).

IPS code what is important in everything we have seen in the past.
IPS - finite time lag

The simplest construction of IPS is based on concentrating on the very recent (finite) past e.g. definite memory machines, finite memory machines, Markov models

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

... 1 2 1 1 2 3 2 1 2 1 2 4 3 2 1 1
... 2 1 1 1 1 3 4 4 4 4 1 4 3 2 1 1
... 3 3 2 2 1 2 1 2 2 1 3 4 3 2 1 1

All three sequences belong to the same IPS “43211”
Markov models (MM)

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 (context-conditional next-symbol distributions)
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
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 $P(\cdot | i)$
Abstract IPS

Things can be much more complicated

May need to remember events that happened long time ago

Time lag into the past can be unbounded
Even more complications ...

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
Finite state skeleton ++

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).
“Pure” state transition formulation

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 $X$ can be considered iterations of a non-autonomous dynamical system on $X$; each map $F_i : X \rightarrow X$ 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,...).
Learning?

Question: To what what degree can such systems be trained on sets of sequences “representing” the underlying language/source structure?

Parametrize the maps $F_i$, $i \in \mathcal{A}$, and fit parameters to the data

Recurrent neural networks (RNN) constitute a popular way of constructing parametric family of maps $F_i$
Recurrent Neural Network (Elman)

Neurons organized in 4 groups: Input (I), Context (C), Recurrent (R) and Output (O)

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

C is an exact copy of R from the previous time step
Activations of neurons in R form IPS of RNN

These activations code the entire history of symbols we have seen so far
RNN - parametrization

\[ \mathbf{R}^{(t)} = f(\mathbf{I}^{(t)}, \mathbf{C}^{(t)}) = f(\mathbf{I}^{(t)}, \mathbf{R}^{(t-1)}) \]

\[ \mathbf{O}^{(t)} = h(\mathbf{R}^{(t)}) \]

\[ R_i^{(t)} = \sigma \left( \sum_j W_{ij}^{RI} \cdot I_j^{(t)} + \sum_j W_{ij}^{RC} \cdot C_j^{(t)} + T_i^R \right) \]

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

\( W^{RI}, W^{RC}, W^{OR} \) 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.
Map $O^{(t)} = h(R^{(t)})$ is a squashed version of a linear transformation, and hence it is quite smooth and monotonic.

IPS $R^{(t)}$ 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.
Knowledge extraction from RNN

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

Each cluster \( i=1,2,\ldots,M \) has associated next-symbol probability distribution \( P( . | i ) \).
The power of contractions
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
RNN prior to training

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
NPM prior to training implement VLMM

NPM prior to training are 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
Chaotic laser

Training sequence: 8000 symbols
Test sequence: 2000 symbols
Results for chaotic laser

(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:
\[ R \rightarrow 1R3, \; R \rightarrow 2R4, \; R \rightarrow e. \]

<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
Results for deep recursion

(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
Theorem (Hammer & Tiño, 2003):

- 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, 2003):**

- 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

Theorem (Tiño & Hammer, 2003):
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.