Text Correction Using Approaches Based on Markovian Architectural Bias

Michal Čerňanský, Matej Makula, Peter Trebatický and Peter Lacko
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3, 812 19 Bratislava
E-mail: {cernansky,makula,trebaticky,lacko}@fiit.stuba.sk

Abstract

Several authors have reported interesting results obtained by using untrained randomly initialized recurrent part of an recurrent neural network (RNN). Instead of long, difficult and often unnecessary adaptation process, dynamics based on fixed point attractors can be rich enough for further exploitation for some tasks. The principle explaining untrained RNN state space structure is called Markovian architectural bias [1, 8] and several methods using this behavior were studied. In this paper we apply these approaches to correct corrupted symbols from symbol sequence. These approaches share some properties with variable length Markov models hence our experiments are inspired by the paper dealing with the text correction on the bible dataset.

1 Introduction

The dynamics of randomly initialized recurrent neural network (RNN) is not "random". On the contrary, its state space shows considerable amount of structural differentiation. It means the activities of recurrent neurons can be grouped in clusters even in an untrained, randomly initialized recurrent neural network [4]. The structure of the state space reflects the history of inputs presented to the network and clusters correspond to the contexts of the variable length Markov models (VLMMs) [6, 7].

The authors have reported promising results working with approaches based on Markovian architectural bias, such as the chaos game representation of input sequence [6] or Echo state networks [2, 3]. Inspired by experiments with VLMM performed in [10] we try to adopt these novel approaches to tasks of practical usage. Specifically, we build predictive models based on chaos game representation (CGR) of the input symbolic sequence and compare their predictive performance to the prediction models based on VLMM. We also apply created predictive models to the text correction task on the bible dataset.

Next section explains the principles and properties of Markovian architectural bias of recurrent neural networks. It is also shown the correspondence to the class of Markov models called variable length Markov models. The third section shows how the predictive models are built and their performance is tested. In the fourth section created predictive models are used on the text correction task and conclusions are formulated in the last section.

2 Architectural Bias of RNNs

RNN training process is considerably more complex in comparison with feed-forward network weight adaptation. Changing recurrent connections in order to obtain desirable dynamics by providing input-output pairs is often difficult and computationally expensive task. The dynamics of a randomly initialized recurrent neural network is surprisingly not "random", instead the RNN state space shows considerable amount of structural differentiation. It means that activities of recurrent neurons even in an untrained, randomly initialized recurrent neural network can be grouped in clusters [4]. The structure of clusters reflects the history of inputs presented to the network. This phenomenon is called Markovian architectural bias and can be explained by means of the iterated function system theory. For example the dynamics of simple RNN can be expressed by equation

$$\mathbf{s}(t) = f\left(\mathbf{W}\cdot\mathbf{s}(t-1) + \mathbf{W^{in}}\cdot\mathbf{i}(t)\right), \qquad (1)$$

where f stands for nonlinear activation function, $\mathbf{W}$ and $\mathbf{W^{in}}$ are matrices of recurrent and input weights respectively. Recurrent and input activities in time t are denoted by $s(t)$ and $i(t)$, respectively. Network output is calculated as

$$\mathbf{o}(t) = f\left(\mathbf{W^{out}}\cdot\mathbf{s}(t)\right), \qquad (2)$$

where $\mathbf{W^{out}}$ is matrix of output weights. Usually nonlinear activation function f is used and recurrent and input weight matrices $\mathbf{W}$ and $\mathbf{W^{in}}$ are initialized to small values drawn

from symmetric interval such as $(-1, 1)$. The notion of Markovian architectural bias can be explained by simplifying RNN dynamics by using linear activation function f and by setting network weights deterministically to:

$$\mathbf{W} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \quad (3)$$

$$\mathbf{W^{in}} = \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \end{pmatrix} \quad (4)$$

Processing of symbol sequence $s_1 s_2 \dots s_t$ created over 4 symbol alphabet $A = \{a, b, c, d\}$ with symbols encoded by one-hot-encoding scheme (if $s_t = a$ than $\mathbf{i}(t) = (1, 0, 0, 0)^T \dots$) would create the following regions in two dimensional state space (Fig. 1).

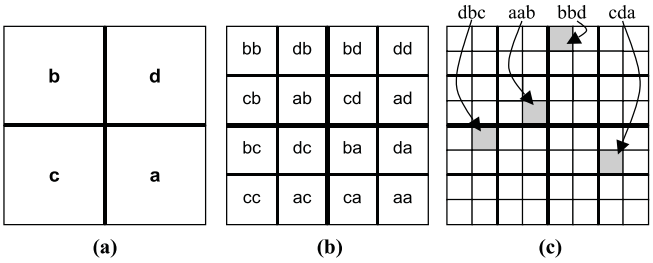

Figure 1. The relationship between the network state and the history of symbols presented to the network with simplified dynamics.

The network state (activities of hidden units) is always located in a region corresponding to the last symbol s_t presented to the network as shown in Fig. 1a. Moreover the position within this region can be further specified by the previously presented symbol s_{t-1}. This "second level" is shown in Fig. 1b and sub-regions differentiated by the symbol presented to the network in time s_{t-2} are shown in Fig. 1c. The position of activities in the network state space is determined by the history of symbols presented to the network with the recent symbols having more important impact than symbols presented in older time steps.

When recurrent and input weights are initialized randomly to small values and activation function f is nonlinear, regions in state space are not positioned precisely and might overlap. However, the main feature of network dynamics – to transform sequences with similar suffix to similar hidden unit activities – is unchanged.

To obtain visual representation of the whole sequence we can insert hidden unit activities $\mathbf{s}(t)$ from all time steps into two-dimensional plot as shown in Fig.2. This way of representing input sequence is called chaos game representation (CGR). The random sequence was created by drawing symbols from alphabet A with equal probability. Laser

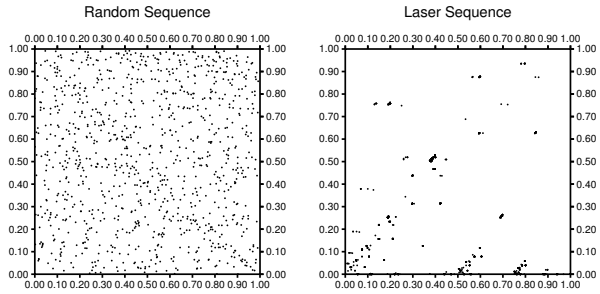

Figure 2. Chaos game representations of sequence of random symbols and Laser sequence (using simplified RNN settings)

data set is the sequence of differences between successive activations of a real laser in chaotic regime [7] which was quantized to symbolic sequence over four-symbol alphabet A. CGR of random sequence results in the state space regularly covered by points. It is not the case of CGR of the laser sequence. Similar subsequences correspond to points that are closer in the state space. The longer the common suffix, the nearer the points are in the state space. Frequent subsequences of longer length produce clusters.

This behavior has led to the idea described in [7] where prediction models called neural prediction machine (NPM) and fractal prediction machine (FPM) were suggested. Both use Markovian dynamics of untrained recurrent network (Eq. 1). In FPM, activation function f is linear and weights are defined deterministically in order to create precise state space dynamics as it was described earlier. In NPM, activation function f is nonlinear and weights are randomly initilized to small values as in regular RNN. Instead of using classical output layer NPM and FPM use prediction model that is created by extracting clusters from the network state space. Each cluster corresponds to different prediction context with the next symbol probabilities.

More precisely, symbol presented to the network drives the network to some state (activities on hidden units). The state belongs to some cluster and the context corresponding to this cluster is used for the prediction. The context's next symbol probabilities are estimated during training process by relating the number of times that the corresponding cluster C is encountered and the given next symbol x is observed N_C^x with the total number when the cluster C is encountered N_C:

$$P(x|C) \approx \frac{N_C^x}{N_C} \quad (5)$$

These models share some properties with variable length Markov models. Building Markov next-symbol prediction model is straightforward by estimating probabilities

$P(x|w) \approx \frac{N_w^x}{N_w}$, where $P(x|w)$ is the probability of symbol $x \in A$ being observed after sequence $w = s_1 s_2 \ldots \in A^*$. In this case sequence of observed symbols w represents the prediction context. In n-th order Markov model the set of contexts is formed of all words of length n which can be created from alphabet A. The set contains $|A|^n$ contexts where $|A|$ is the size of the alphabet.

Variable length Markov models (VLMMs) [5, 10] contain contexts with different lengths. They try to evaluate the context importance and irrelevant contexts are not kept by the model. The importance of the context is estimated from the training sequence. There is a natural idea of growing of VLMMs. Existing context $w \in A^*$ is extended by symbol $x \in A$ and new context xw is added to the model if its next symbol probability distribution $P(.|xw)$ significantly differs from the next symbol probability distribution of original context $P(.|w)$ and its occurrence $P(xw)$ is important enough. A natural way of representing VLMMs is in the form of a prediction suffix trees (PSTs). Fig. 3 shows simple PST created for laser sequence. For example the probability of observing symbol b after the sequence $ccca$ will be estimated as $P(b|ccca) = 0.06$.

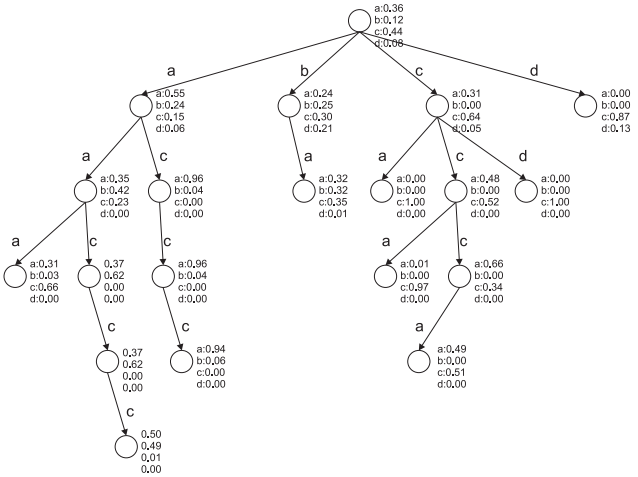

Figure 3. Simple variable length Markov model represented by the prediction suffix tree for the symbolic laser sequence.

We can graphically represent the regions of CGR of symbolic laser sequence created using simplified RNN dynamics (Fig. 2b) that correspond to the contexts of variable length Markov model created for symbolic laser sequence (Fig. 3). In Fig. 4 the shades of gray are used to indicate the corresponding VLMM context. Longer contexts correspond to the smaller regions and the darker shades of gray are used. As already mentioned, the clusters in the RNN state space correspond to the frequent and similar (hence from the modeling point of view important) substrings of sequence presented to the network. As can be seen in Fig. 3 the clusters in the state space of recurrent neural networks with simplified dynamics correspond to the prediction contexts of variable length Markov models. The structure of RNN state space initialized with small weights corresponds to the prediction contexts of Markov model hence the name for Markovian architectural bias.

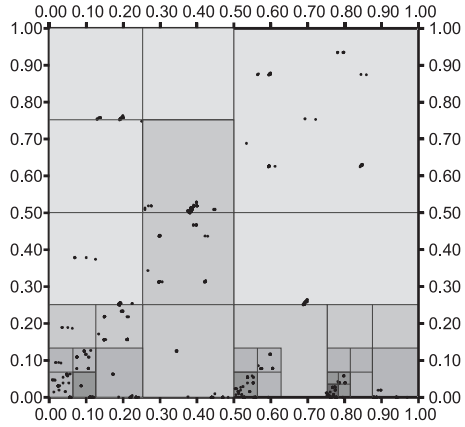

Figure 4. Chaos game representation of the symbolic laser sequence that correspond to the contexts of variable length Markov model created for symbolic laser sequence.

3 Creating Prediction Models

The purpose of this paper is to show how approaches based on Markovian architectural bias can be used for symbolic sequence processing. In [10] the authors used variable length Markov model on the bible dataset. Since approaches based on the chaos game representation (FPM, NPM) share some properties with VLMM we decide to perform experiments with the same dataset. We modified english version of King James Bible and took out the Genesis book as test sequence. The rest of books served as a training sequence. The length of test sequence was $1.9 \cdot 10^5$ and the length of training sequence was $2.9 \cdot 10^6$. Alphabet includes 26 characters and a space character.

3.1 Prediction Model Based on VLMM

We repeated experiments with VLMM as in [10]. The error criteria based on Kullback-Leiblerg divergence was used to evaluate context importance.

$$Err(xw, w) = P(xw) \sum_{a \in A} P(a|xw) log \frac{P(a|xw)}{P(a|w)}, \quad (6)$$

where xw is a candidate context associated with a child node and w is a context associated with parent node in the prediction suffix tree being iteratively built. If statistical difference was superior to chosen ratio ϵ, the child node with context xw was added to the model. All nodes up to the specified length L were tested. Laplace correction was used to avoid estimating any probabilities of zero for events never observed in training data. Next symbol probabilities were estimated using:

$$P(w) \approx \frac{\gamma + N_w}{\gamma|A| + \sum_{v \in A^{|w|}} N_v}, \quad (7)$$

$$P(x|w) \approx \frac{\gamma + N_w^x}{\gamma|A| + \sum_{a \in A} N_w^a}, \quad (8)$$

where N_v is a number of occurrences of string v in a sample sequence and N_v^x is a number of observations of symbol x after string v. Laplace correction parameter γ was set to $|A|^{-1}$.

Training prediction suffix tree is summarized in the following code in pseudo-language:

```
ALPHABET                  - sequence alphabet, a set of symbols
StrLength(W)              - the length of string W
L                         - maximum tree depth
Eps                       - minimum statistical surprise
Error(W)                  - calculates statistical surprise of string w

CSTRINGS                  - set of candidate strings
StringsInit(STRS)         - initialize set of strings STRS
StringsInsert(STRS,W)     - insert string W into set of strings STRS
StringsRemove(STRS)       - removes string from set of strings STRS
StringsIsEmpty(STRS)      - test set of strings STRS for being empty

PST                       - prediction suffix tree
TreeInit(TREE)            - initialize tree TREE
TreeInsert(TREE,W)        - insert context W into tree TREE
```

Algorithm description:

First, the set of candidate strings is initialized with one symbol words formed of symbols from alphabet (lines 1 to 3). Then the prediction suffix tree PST is initialized with only root node corresponding to the empty string e. The next symbol probability table is also calculated (line 5). The tree is iteratively built in the loop (lines 7 to 15). If the set of candidate strings is not empty (line 7), a candidate string is removed from the set (line 9), the statistical surprise of the string is evaluated and if relevant enough (line 10) the string is inserted into the tree PST (line 11). All ancestor strings not present in the tree are also inserted and the next symbol probabilities are calculated for all strings being inserted. If maximal tree depth was not reached (line 12)

```
1  StringsInit(CSTRINGS);
2  foreach A from ALPHABET do
3    StringsInsert(CSTRINGS, A);
4
5  TreeInit(PST);
6
7  while NOT StringsIsEmpty(CSTRINGS) do
8  begin
9    CSTRING := StringsRemove(CSTRINGS);
10   if Error(CSTRING) >= Eps then
11     TreeInsert(PST, CSTRING);
12   if StrLength(CSTRING) < L then
13     foreach A from ALPHABET do
14       StringsInsert(CSTRINGS, A+CSTRING);
15 end;
```

Figure 5. Algorithm for training prediction suffix tree

the set of candidate strings is enhanced with novel strings formed by concatenation of symbols of alphabet with the processed string (lines 13 and 14). Operator + is meant as a concatenation of the strings.

3.2 Prediction Model Based on Chaos Game Representation

Models such as FPM and NPM are based on attractor dynamics of RNN-like state space. Prediction contexts correspond to clusters in the state space driven by equation 1. For both NPM and FPM one-hot encoding of input alphabet was used resulting in input layer containing $|A| = 27$ units.

FPM models had deterministically set recurrent weights and linear activation function was used. We investigated two possibilities of initializing network dynamics of FPM. The first model labeled FPM5 was composed of $\lceil log_2(|A|) \rceil = 5$ recurrent units, recurrent weights were initalized as $\mathbf{W} = k \cdot \mathbf{I}$ where $k = 0.5$ is the contraction ratio. Columns of input weight matrix $\mathbf{W^{inp}}$ are binary displacement vectors multiplied by $1 - k$ in the same manner as described in eq. 4. The second model labeled FPM27 use $|A| = 27$ recurrent units and weight matrices were initialized to $\mathbf{W} = k \cdot \mathbf{I}$ and $\mathbf{W^{inp}} = (1 - k) \cdot \mathbf{I}$.

We have also performed several experiments with NPM models. Matrices $\mathbf{W}$ and $\mathbf{W^{inp}}$ were randomly initialized from symmetric interval $(-1, 1)$ and sigmoidal activation function was used. Dynamics driven by randomly set weights combined with nonlinear transfer function work remarkably well in case of ESN models, to our knowledge no deterministic method was proposed that would perform as well. Similarly to the FPM models, we used NPM5 and NPM27 models having 5 and 27 hidden units, respectively.

Vector quantization techniques such as popular K-means quantization can be used for identifying clusters. Slow convergence is the major drawback of K-means which limits

its practical applicability to larger datasets. To make training process feasible and at least partially comparable to the VLMM training we estimated clusters only on the fraction of activities produced by models. Overall training process can be summarized in several simple steps:

1. Activation Creation
   By running through training sequence $S = s_1 s_2 s_3 \dots s_T$ symbols s_t are presented to the network and activities on hidden units ($\mathbf{s}(t)$ from eq. 1) are recorded into sequence of activities $X = (X_1, X_2, X_3 \dots X_T)$. Dimension of vectors X_t is given by the hidden layer size and was 5 for FPM5 and NPM5 and 27 for FPM27 and NPM27.

2. Quantization
   For a given number m of prediction contexts clusters representing activities X are found. K-means quantization was performed on a fraction of X (every 100th activity), resulting in m centers $C = (C_1, C_2, \dots C_m)$ representing clusters.

3. Creating the Next Symbol Predictions Probabilities
   The next symbol counter $N^x_{C_i}$ is created for each cluster i represented by center C_i and for each symbol x from alphabet A. By running through training sequence S and sequence of activities X with index $0 < t < n-1$ cluster C_i corresponding to the activity X_t is found and the next symbol counter $N^{s_{t+1}}_{C_i}$ associated with the next symbol observed s_{t+1} is incremented. The next symbol probabilities associated with clusters are estimated as:

$$P(x|C_i) \approx \frac{\gamma + N^x_{C_i}}{\gamma|A| + \sum_{a \in A} N^a_{C_i}}, \qquad (9)$$

   where Laplace correction parameter γ was set to $|A|^{-1}$.

3.3 Experimental Results

Predictive performance was evaluated by means of a normalized negative log-likelihood (NNL) calculated over the test symbol sequence $S = s_1 s_2 \dots s_T$ from time step $t = 1$ to T as

$$NNL = -\frac{1}{T} \sum_{t=1}^{T} \log_{|A|} P_t(s_t), \qquad (10)$$

where the base of the logarithm is the alphabet size, and the $P_t(s_t)$ is the probability of predicting symbol s_t in the time step t. For VLMM the value $P_t(s_t)$ is equal to the $P(s|w)$ where w is the longest suffix of $s_1 \dots s_{t-1}$ found in the PST. For FPM and NPM the testing procedure resembles the training process steps:

1. Activation Creation
   By running through test sequence $S = s_1 s_2 s_3 \dots s_T$ symbols s_t are presented to the network and activities on hidden units are recorded into sequence of activities $X = (X_1, X_2, X_3 \dots X_T)$.

2. Finding the Next Symbol Predictions Probabilities
   Probabilities for NNL calculations $P_t(s_t)$ equal $P(x|C_j)$ where x is the next observed symbol $x = s_t$ and C_j is the center of cluster j representing the previous activity X_{t-1} (C_j is the nearest center from all m centers).

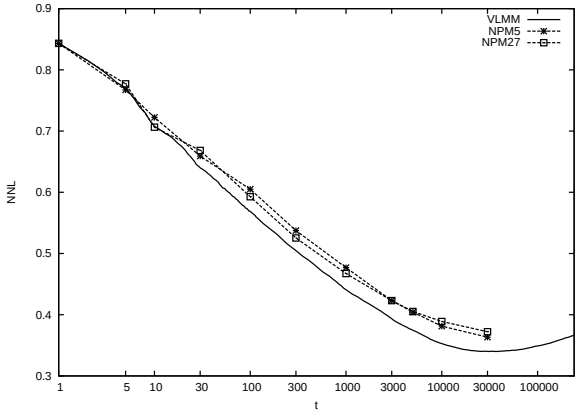

Figure 6. The next symbol prediction performance for VLMM and NPM models.

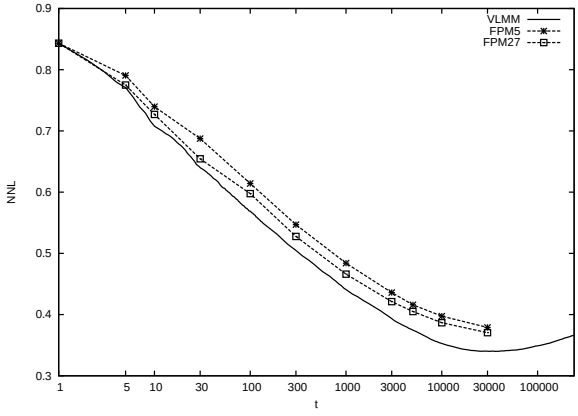

Figure 7. The next symbol prediction performance for VLMM and FPM models.

Prediction models with different number of clusters ranging from 1 to 30000 were created for FPM and NPM.

Single VLMM was created with statistical surprise parameter set to very small value of $\epsilon = 0.000001$ and maximal context length (maximal PST depth) set to $L = 30$. Final suffix tree contained more than 200000 nodes and resulting predictive performance on the test sequence was about $NNL = 0.366$. Modifying PST by removing the least important leaves in statistical surprise sense resulted in VLMMs having from about 200000 to 1 node, hence predictive performances of these models can be shown as a smooth line in figures Fig. 6 and Fig. 7. Predictive performance of models based on the chaos game representation is slightly worse but comparable with VLMM. Both NPM5 and NPM27 have similar performance (Fig. 6). In the case of FPM models (Fig. 7) enhancing the model state space seems to give slightly better results, but in general the state space enhanced to more dimensions did not prove to be useful as it is in the case of the bible dataset.

4 Text Correction

Much of the effort in building recognition systems is devoted to correct the corrupted sequences. In many optical and handwriting character recognition systems, the last stage employs natural-language analysis techniques to correct corrupted sequences.

We chose the problem of correcting the corrupted text in order to compare performance of proposed techniques on the real-world problem. These techniques are simple and efficient and yet provide even better results than soffisticated methods using huge corpora and dictionaries [10].

4.1 Transforming PST into Probabilistic Finite Automaton

Before the text correcting technique can be applied, it is neccessary to transform prediction suffix tree into probabilistic finite automaton (PFA). To be able to achieve this, all the internal nodes of PST have to be of full degree and for every leaf s, the longest prefix of s must be either leaf or an internal node of resulting PST. When adding new nodes to the tree in order to fulfill this requirement, these nodes inherit their next symbol probability function from their respective parents. The states of the PFA then correspond to the leaves of resulting PST including their next symbol probability functions [10].

One also has to compute the initial probability distribution over the states of PFA. Sum of initial probabilities of all states must be 1. The initial probability of each state equals the sum of products of probability of transfering to that state and the initial probability of state from which we transfer from. Thus one obtains $n + 1$ equations in n unknowns.

4.2 Extracting PFA from CGR State Space

The state space of CGR can be successfully used for the prediction model creation as described in the previous section using process consisting of the clusterization of unit activations and the estimation of the next symbol probability distribution for found clusters. Very similar process can be used for creating a probabilistic finite automaton based on the CGR state space. The PFA creation process consists of 4 steps, the first two steps were already described in the previous section:

1. Activation Creation
2. Quantization
3. Creating the State Transition Probabilities
   Clusters found in the quantization step correspond to the PFA states. The state transition counter $N_{C_i}^{C_j}$ is created for each pair of states C_i and C_j. By running through sequence of activities X with index $0 < t < n-1$ clusters C_i and C_j corresponding to the activities X_t and X_{t+1} are found and the state transition counter $N_{C_i}^{C_j}$ is incremented. Finally the state transition probabilities associated with clusters are estimated as:
   $$P(C_j|C_i) \approx \frac{N_{C_i}^{C_j}}{\sum_k N_{C_i}^{C_k}} \tag{11}$$
4. Creating the Observation Symbol Probabilities
   The observation symbol counter $N_{C_i}^{x}$ is created for each PFA state i represented by center C_i and for each symbol x from alphabet A. By running through training sequence S and sequence of activities X with index $0 < t < n$ cluster C_i corresponding to the activity X_t is found and its observation symbol counter $N_{C_i}^{s_t}$ associated with the symbol s_t is incremented. The probabilities of observing symbol x when being in state C_i are estimated as:
   $$P(x|C_i) \approx \frac{\gamma + N_{C_i}^{x}}{\gamma|A| + \sum_{a \in A} N_{C_i}^{a}}, \tag{12}$$
   where Laplace correction parameter γ was set to $|A|^{-1}$.

The last step of creating observation symbol probabilities is almost identical to the process of finding the next symbol probabilities for the prediction model as described in the previous section. Although probability distributions over all possible symbols are considered, in great majority of cases only one symbol is strongly predominant respecting the principles of architectural bias. For PFA having significantly more states than the number of symbols in alphabet only this predominant symbol can be kept with the state

and the version of Viterbi algorithm described later can be used for the text correction. Similar approach of extracting stochastic machines from RNN was desribed in [9].

4.3 Viterbi Algorithm

The task of repairing corrupted text (sequence of symbols) means to find the most likely state sequence that generated the original text [10]. We assume that the text was created by the same stochastic process which created training data and that the corrupting independent identically distributed noise probability is known.

Then we can solve this task using a well known dynamic programming algorithm in signal processing called Viterbi algorithm [11]. The algorithm is presented by the following code in pseudo-language:

`SEQUENCE`	- corrupted sequence of symbols
`STATES`	- set of states of PFA
`PS`	- vector of probabilities of being in `i`'s state
`PSN`	- newly constructed vector `PS`
`ErrorMatrix`	- probability of changing symbol `i` to `j`
`Transitions(T)`	- set of transitions from state `T`
`NextState(N)`	- next state corresponding to transition `N`
`Symbol(N)`	- symbol corresponding to transition `N`
`Prob(N)`	- probability of transition `N`
`TabPrev`	- previous state for state `i` and `j`'s symbol in sequence

```
1  PS := initial probabilities of states
2
3  foreach Sj from SEQUENCE do
4  begin
5    PSN := Zeros;
6    foreach T from STATES do
7    begin
8      foreach N from Transitions(T) do
9      begin
10       K := NextState(N);
11       P := ErrorMatrix [Sj, Symbol(N)];
12       if PSN [K] < P * PS [T] * Prob(N) then
13       begin
14         PSN [K] := P * PS [T] * Prob(N);
15         TabPrev [K,j] := T;
16       end;
17     end;
18   end;
19   PS := PSN;
20 end;
21
22 ReconstructCorrectedSequence(TabPrev);
```

Figure 8. Viterbi algorithm for text correction

In order to reconstruct the corrected sequence one has to find maximum value in the vector `PS`. Thus we obtain the state in which the most likely state sequence we are looking for ends. Previous state is in the `TabPrev[K,j]`, where `K` is our most recently obtained state and `j` the position of symbol in the sequence we are correcting. This step is repeated for every symbol from the sequence, going from end.

Now that we have the most likely state sequence, we take the last symbol from context of each state, thus obtaining the repaired symbol sequence.

The overall complexity of this algorithm is $O(NTS)$, where N is the number of symbols in alphabet, T number of states and S length of sequence to be corrected.

4.4 Text Correction Using Greedy Search

Models based on chaos game representation have more complicated dynamics comparing to VLMM and some information might be lost during conversion to PFA. This is due to the fact that NPM and FPM are not finite state predictors and their dynamics is completely equal to the dynamics of original underlying neural network. The most likely state sequence can therefore be found only by examining probabilities of all possible 27^T state trajectories.

To find solution in reasonable time we have decided to reduce search space by Greedy search (GS), which trace only limited number of most probable candidate trajectories. At every step all possible continuations of candidate trajectories were examined and only the most likely continuations were taken to the next step. Probability of state trajectory was calculated similarly to Viterbi algorithm, while taking into account both initial probability of state trajectory and probability of transition with respect to the observation. Finally, the most probable state trajectory was chosen as a winner and original sequence was reconstructed.

4.5 Experimental Results

To test the text correction capabilities of VLMM the prediction suffix tree was constructed using the learning algorithm on training part of the bible with parameters $\epsilon = 0.0001$ and $L = 30$, resulting in PST having 1790 nodes. PST was then transformed into PFA with 13911 states. Text correction capabilities of models based on chaos game representation were tested using NPM5 with 30000 prediction contexts. Input and recurrent weights of underlying network were initialized to small values drawn from symmetric interval $(-1, 1)$. Clusters from network state space were extracted by K-means clustering algorithm. Text correction was performed by either greedy search of 100 most probable state trajectories or by Viterbi algorithm applied to PFA extracted from NPM5 state space.

We tested the text correcting algorithm on text of length 1000 taken from the beginning of the book Genesis. We changed each symbol into some other one from alphabet with probability 0.1. As a measure we took the ratio of

corrected symbols to originally corrupted ones, as well as the number of corrupted symbols still remaining in the corrected text, because the process can also corrupt some originally uncorrupted symbols.

We performed 10 such experiments and took average of both measures. The results show that the VLMM successfully corrected 77.7% of originally corrupted 100.4 symbols and that 26.8 symbols still remained in the corrected text. The greedy search applied on CGR successfully corrected 85.2% of originally corrupted 98.6 symbols, while 20.2 of misclassified symbols still remained in the corrected text. The best results were achieved by PFA extracted from CGR state space with correction ratio of 87.5%. Generally, the majority of errors was removed, but some remained uncorrected and some were introduced. This is due to the fact e.g. that the process found another word that was more probable than the original uncorrupted one.

	corrupted	corrected	ratio	remaining
VLMM / PFA	100.4	78.0	0.7769	26.8
CGR / PFA	101.9	88.8	0.8748	18.3
CGR / GS	98.6	83.8	0.8516	20.2

5 Conclusion

In our work we have studied approaches based on Markovian architectural bias for modeling of symbolic sequences. These approaches use multidimensional RNN-like state space with contractive dynamics that serves as the base for the creation of probabilistic models. Models with dynamics created randomly (NPM) and deterministically (FPM) were compared with finite state VLMM predictor. First the predictive performance was evaluated and compared. Created models were then used in application of the text correction.

All approaches gave similar results on the task of the next symbol prediction, although VLMM slightly outperforms CGR based models. The key part of CGR model creation is the quantization step that is computationally demanding and significantly limits overall model performance. VLMM based approach was compared with two techniques based on CGR on the text correction task. The VLMM prediction suffix tree was converted into PFA and the text was corrected using Viterbi algorithm. The first technique based on CGR consisted of extracting PFA from CGR representation and application of classical Viterbi algorithm for text correction.

The second approach directly uses the state space dynamics of CGR representation and original text was reconstructed by greedy version of Viterbi algorithm. The correction performance of CGR models was better since they could be based on better predictive models, because the process of converting VLMM prediction suffix tree to PFA results in significant increase in PFA states (in comparison to the PST nodes). In the case of CGR models the conversion was straightforward (CGR/PFA) or not needed at all (CGR/GS).

The main drawback of models based on the CGR, that can significantly reduce their applicability, is the high computational requirements mostly due to the quantization step. Various dynamic programming techniques significantly reduce computational requirements when working with PST. Other quantization methods e.g. based on hierarchical clustering or other ways of exploitation of RNN-like state space may improve model quality.

Acknowledgment

This work was supported by the grants APVT-20-030204 and APVT-20-002504.

References

[1] M. Christiansen and N. Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:417–437, 1999.

[2] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[3] H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[4] J. F. Kolen. The origin of clusters in recurrent neural network state space. In *Proceedings from the Sixteenth Annual Conference of the Cognitive Science Society*, pages 508–513. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

[5] M. Machler and P. Buhlmann. Variable length markov chains: methodology, computing and software. *Journal of Computational and Graphical Statistics*, 13:435–455, 2004.

[6] P. Tiňo. Spatial representation of symbolic sequences through iterative function system. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, 29(4):386–392, 1999.

[7] P. Tiňo and G. Dorffner. Recurrent neural networks with iterated function systems dynamics. In *International ICSC/IFAC Symposium on Neural Computation*, 1998.

[8] P. Tiňo, M. Čerňanský, and L. Beňušková. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1):6–15, 2004.

[9] P. Tiňo and V. Vojtek. Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World*, 8(5):517–530, 1998.

[10] D. Ron, Y. Singer, and N. Tishby. The power of amnesia. *Machine Learning*, 25, 1996.

[11] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.