Quantifying Long Range Dependence in Language and User Behavior to improve RNNs

Francois Belletti
belletti@google.com
Google Research
Mountain View, CA

Minmin Chen
minminc@google.com
Google Research
Mountain View, CA

Ed H. Chi
edchi@google.com
Google Research
Mountain View, CA

ABSTRACT

Characterizing temporal dependence patterns is a critical step in understanding the statistical properties of sequential data. Long Range Dependence (LRD) — referring to long-range correlations decaying as a power law rather than exponentially w.r.t. distance — demands a different set of tools for modeling the underlying dynamics of the sequential data. While it has been widely conjectured that LRD is present in language modeling and sequential recommendation, the amount of LRD in the corresponding sequential datasets has not yet been quantified in a scalable and model-independent manner. We propose a principled estimation procedure of LRD in sequential datasets based on established LRD theory for real-valued time series and apply it to sequences of symbols with million-item-scale dictionaries. In our measurements, the procedure estimates reliably the LRD in the behavior of users as they write Wikipedia articles and as they interact with YouTube. We further show that measuring LRD better informs modeling decisions in particular for RNNs whose ability to capture LRD is still an active area of research. The quantitative measure informs new Evolutive Recurrent Neural Networks (EvolutiveRNNs) designs, leading to state-of-the-art results on language understanding and sequential recommendation tasks at a fraction of the computational cost.

CCS CONCEPTS

• **Mathematics of computing** → **Time series analysis**; • **Information systems** → *Personalization*; *Language models*; • **Computing methodologies** → *Learning latent representations.*

KEYWORDS

data mining; sequential learning; long range dependence; RNN

ACM Reference Format:
Francois Belletti, Minmin Chen, and Ed H. Chi. 2019. Quantifying Long Range Dependence in Language and User Behavior to improve RNNs. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330944

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08.
https://doi.org/10.1145/3292500.3330944

1 INTRODUCTION

Sequential modeling based on state-of-the-art Machine Learning (ML) techniques such as RNNs has proven very successful for applications such as natural language processing [2, 3, 11, 32, 46], machine translation [11], speech recognition [18], and recommender systems [4, 22, 44, 52]. As sequential modeling uses previously observed data to predict future states and observations, the rate at which past information becomes less relevant to present dynamics is a key aspect of the data. LRD focuses on sequences where past information losses its influence with a "power-law" decay — as opposed to exponential — which corresponds to many real-world time series (in finance, network traffic prediction, behavioral analysis [38], human genomes, musics and languages). The presence of LRD in sequential data dictates modeling choices, as higher LRD implies that perturbations have a longer lasting footprint which requires sequential models with longer memories.

While principled theories of LRD exist to analyze datasets, build models and prove limit theorems for real-valued time series [6, 17, 38, 42], the counterparts for sequences of discrete symbols are not readily available [4, 26, 47], especially when the set of symbol values (e.g. number of words in the English language) is large. Being able to quantify the amount of LRD in such sequences can help better understand the statistical properties of the sequential dynamics and inform modeling decisions. For instance, a more precise understanding of temporal dependencies in the data can guide the architectural design for language understanding and recommender systems — two major areas of application of ML in production systems — that has to conciliate two seemingly antithetic goals: leveraging long term history to better inform the next prediction and serving the prediction under a short latency deadline.

RNNs constitute the current state-of-the-art solution for sequential language and user behavior modeling [4, 8, 10, 16, 35, 39] and are widely employed in production systems. RNNs are parametric non-linear models defined by a recurrent equation involving real-valued vectors:

$$[\hat{Y}_t, M_t]^T = \Phi_\theta (X_t, M_{t-1}) \tag{1}$$

where X_t and $\hat{Y}_t$ are the input and output at each timestep t and M_t is the state of the RNN storing past information. The ability of such networks to capture LRD patterns remains the subject of active research, with a large body of work focusing on improving the trainability of RNNs [4, 10, 35, 36, 48] in the LRD setting.

Enabling LRD in sequential neural models has been actively researched, in particular for language understanding [8, 10, 35] and sequential recommendations [4, 16, 39, 47], but remains challenging. Better accounting for long term user memory provides better behavioral predictions however LRD models are often difficult to

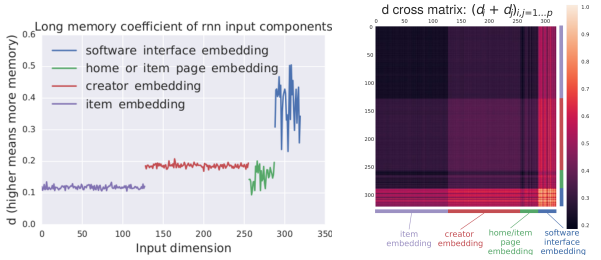

Figure 1: Estimating Long Range Dependence (LRD) in user/item interactions on YouTube: we measure the LRD coefficient (i.e. statistical sequential memory) in user behavior. The estimated LRD coefficients correspond to four groups of symbol embeddings. We present univariate memory coefficient (top) and covariate memory coefficients (bottom). For a large distance h, the component i, j of the auto-covariance matrix $\gamma_X(h)$ decays proportionally to $h^{d_i+d_j-1}$. A higher coefficient d means that a higher amount of memory is present and d significantly higher than 0 demonstrates that the time series under consideration is Long Range Dependent. The estimates are obtained by applying the log-periodogram regression technique to sequences of symbols translated into their embeddings.

serve in latency-sensitive production environments and at the scale of an entire online platform. We show in the following that a better quantitative measurement of LRD leads to a finer understanding of behavioral dynamics which in turn enables more subtle trade-offs between model expressiveness and computational cost. The present paper improves the understanding of LRD in sequential behavioral modeling as follows:

(1) We provide an estimation method for LRD patterns in sequences of symbols belonging to large vocabularies, typical of language modeling and sequential recommendation tasks. To the best of our knowledge, we are the first to offer a model-free estimation of such dependencies;
(2) We show how the quantitative insights we extract can be turned into design decisions for RNNs operating under tight latency constraints. We propose a class of RNNs named Evolutive RNNs (EvoRNNs) which vary model capacity over time to match the sequential dependency patterns;
(3) We demonstrate the computational efficiency as well as quality gain of EvolutiveRNNs on the Billion Word language modeling task and an anonymized proprietary data set used to improve sequential recommendations for YouTube.

2 RELATED WORK

2.1 LRD in real-valued time series

We follow the presentation of LRD given in [38] and consider a second-order stationary real-valued time series $(X_t)_{t\in\mathbb{Z}}$, abbreviated to (X). By assumption, the mean of the time series $\mu_X = E[X_t]$ and its auto-covariance function $\gamma_X(h) = Cov(X_t, X_{t+h})$ are well defined and do not change over time. The spectral density f_X of (X) is also well defined and verifies $\int_{-\pi}^{\pi} f_X(\lambda)e^{ih\lambda}d\lambda = \gamma_X(h)$. Prior to delving in the topic of LRD, we recall the definition of slow varying functions, *e.g.*, the logarithm.

Definition 1. ***Slow varying function:*** *A function L is slow varying at infinity if it is positive on some interval $[c, \infty)$ where $c \geq 0$ and for any $a > 0$*

$$\lim_{u\to\infty} \frac{L(au)}{L(u)} = 1.$$

A function L is slow varying near 0 if $u \to L(\frac{1}{u})$ is slow varying at infinity.

Five different non-equivalent definitions of LRD are given in [38]. Here we only consider two equivalent definitions given respectively in the time and frequency domain.

Definition 2. ***Long Range Dependent (LRD) mono-variate real-valued time series:*** *The mono-variate real-valued time series (X) is LRD iff. there exists a real $d \in (0, \frac{1}{2})$, referred to as the **LRD coefficient of** (X), such that*

$$\gamma_X(h) = L_\infty(h)h^{2d-1}, \text{ or equivalently } f_X(\lambda) = L_{0^+}(\lambda)\lambda^{-2d},$$

where L_∞ and L_{0^+} are slow varying functions at infinity and 0 respectively.

Higher value of d indicates a slower decay of temporal dependence and therefore a higher amount of memory in the time series. Some readers may be acquainted with the Hurst index H which measures the amount of LRD in a stochastic process through its self-similarity and scaling properties [30, 38, 45]. If $H \in (\frac{1}{2}, 1)$ then H verifies $d = H - \frac{1}{2}$ (this property is for instance proven for Fractional Brownian Motions in [38]).

A consequence of the time series (X) being LRD is that the variance of $\frac{\sum_{t=1}^{N} X_t}{N}$ decays much slower as $L_\infty(N)N^{2d-1}$ where L_∞ is another slow varying function at infinity. LRD is indeed notorious for changing convergence rates of M estimators as compared to the case of *iid.* observations [6, 17, 38, 42].

As explained in [6, 17, 38, 42] there are multiple standard estimators for the LRD coefficient d such as the Rescaled Range estimator R/S, wavelet based, and variance estimation based estimator. Maximum Likelihood Estimators for generative linear LRD models such as FARIMA models are also available. One long-standing estimator for d is the log-periodogram estimator [41] which focuses on the spectral density $f_X(\lambda) = L_{0^+}(\lambda)\lambda^{-2d}$. Let $\widehat{f_X(\lambda)} \equiv |FFT_N[\lambda](X)|^2 = |\sum_{t=1}^{N} X_t e^{-it\lambda}|^2$ denote the empirical spectrum of (X) — assuming N observations of the time series are available — then one can measure d through the estimation of the slope $b = -2d$ in the affine relationship

$$\log\left(\widehat{f_X(\lambda)}\right) = a + b\log(\lambda) \tag{2}$$

by ordinary least squares regression in the domain of low frequencies [41].

Although Maximum Likelihood Estimation is now preferred for measuring LRD in time series [38], we employ the log-periodogram estimate here to avoid assuming a particular generative model for the data. Therefore, we propose to methodically quantify LRD in sequences of symbols in large vocabularies/inventories through the spectral density of sequences.

2.2 LRD in sequences of symbols

LRD often manifests itself in physical and societal phenomena through a slow decay of temporal dependence which is usually observed in the form of a power-law decaying auto-covariance

function [38]. Per Definition 2, this time domain power-law decay at infinity is equivalent to a power-law divergence of the spectral density in the frequency domain near 0.

LRD estimation has become standard in the study of real-valued time series and has led to improvements in LRD predictions or risk assessment thanks to models such as FARIMA [38, 45]. In contrast, while it is widely assumed that LRD is a key feature of the input sequences that needs to be captured for better predictions in language modeling and sequential recommendations, the amount of LRD in these tasks remains to be estimated in a principled manner.

A key difference between real-valued time series and language modeling or sequential recommendation tasks is that the latter generally involve sequences of discrete symbols or items from vocabularies of 10^5 to 10^7 distinct values. For small vocabularies of symbols, computing the decay of mutual information along the time axis helps quantify the amount of LRD as in sequences of characters [26]. Unfortunately, these techniques do not scale to large vocabularies due to sparse observations and the prohibitively large number of possible combinations. In the present paper we show how alternate representations of symbols can scale estimates of the LRD coefficient d to sequences involving large vocabularies of symbols.

2.3 Gradient propagation and LRD in RNNs

Model-free estimators of LRD such as the log-periodogram estimator differ radically from usual measures of LRD employed in sequential neural models. A substantial body of work concerned with the application of RNNs to LRD sequences of inputs focuses on the propagation of gradients through time.

From the seminal paper on the difficulty of training RNNs [36] to recent developments [4, 35], exploding or vanishing gradients are considered the main obstacle to LRD modeling in RNNs. Various approaches have been proposed to address the issues. Modifications started by introducing gating as in LSTMs [23] and GRUs [13] and later by building multi-scale temporal structure [8, 12], constraining on the spectrum of learned parameter matrices [1, 24, 51], regularization [31, 48] and initialization schemes [10] to improve the trainability of RNNs. It is worth mentioning here that RNNs are not the only neural models to provide good performance with long sequences of inputs. For instance, dilated convolutional architectures [49, 55] have been offered as an effective alternative. Attention is also readily able to capture LRD patterns as part of Transformer [50] but unfortunately it is challenging to serve such a multi-layer attention network with the very low latency required by recommender systems. An alternate solution may be to use a single attention layer as part of a Mixture-of-Experts [47].

The key novelty of our approach is to not measure LRD as the propagation of information through a RNN but rather estimate LRD in the sequences of inputs themselves and design model architectures to match the dependence patterns. Therefore, although we mostly focus on architectural insights for RNNs, our approach is in no way limited to this class of models and could inform the design of other sequential models such as convolution or attention [2] based neural architectures.

3 ESTIMATION OF LRD WITH DISTRIBUTED WORD AND ITEM REPRESENTATIONS

The lack of model-independent measures of LRD blurs the distinction between the amount of LRD present in input sequences and the ability of neural sequential models to capture such LRD. To fill this lacuna, we now develop an estimate for the LRD coefficient d as defined in Definition 2 for long sequences of symbols with large vocabularies, which are common in language modeling and sequential recommendation tasks.

3.1 From sequences of discrete symbols to real valued embedding time series

For tasks involving sequences of symbols belonging to large vocabularies (10^4 to 10^7 in size), it has become standard practice to embed each item in the form of learned vector of real values since the seminal work on word2vec [33, 34] and Glove embedding [37].

These methods map discrete symbols to real-valued vectors in $\mathbb{R}^p$. In practice, a few hundred embedding dimensions are sufficient to provide state-of-the-art predictive performance for tasks with vocabularies of several millions of symbols [4, 15, 33]. Close examination of the inter-item relationships inherited from these continuous representations [28, 33, 34, 53] suggests that related items are indeed collocated in the embedding space.

With these embeddings, we can map sequences of discrete symbols to sequences of real-valued vectors, and use methods developed for real-valued multi-variate time series for analysis. In particular, the well established theory of LRD [38] can be used to characterize the sequential dependence properties of sequences of learned item embeddings. While most existing work focuses on interpreting [33, 34], assessing [53], and visualizing [28] inter-item relationships, to the best of our knowledge, we are the first to examine such relationships *longitudinally along the time axis*.

3.2 Estimation methods for LRD

Although we have exposed the definition of the LRD coefficient d in the mono-variate setting, we still need to extend the presentation to multi-variate time series as item embeddings are real-valued vectors. Here we again follow the presentation given in [38].

Consider a multi-variate second-order stationary time-series (X_t) with $X_t \in \mathbb{R}^p$. We denote $\gamma_X(h) = Cov(X_t, X_{t+h})$ the matrix-valued auto-covariance function of (X) which takes values in $\mathbb{R}^{p\times p}$ and f_X the corresponding spectral density matrix which also takes values in $\mathbb{R}^{p\times p}$. By definition γ_X and f_X satisfy $\forall j,k \in 1\dots p,\ \forall h \in \mathbb{Z}$,

$$\gamma_{X,j,k}(h) = \int_{-\pi}^{\pi} f_{X,j,k}(\lambda) e^{ih\lambda} d\lambda.$$

Definition 3. ***Long Range Dependent (LRD) multi-variate real-valued time series:*** *The multi-variate real-valued time series $(X_t)_{t\in\mathbb{Z}}$ is LRD iff. there exists a real vector $(d_i)_{i=1\dots p} \in (0, \frac{1}{2})^p$ such that*

$$\gamma_{X,j,k}(h)(\lambda) = L^{j,k}_{+\infty}(h)\ h^{d_j+d_k-1}$$

or equivalently

$$f_{X,j,k}(\lambda) = L^{j,k}_{0+}(\lambda)\ \lambda^{-(d_j+d_k)}$$

where $L^{j,k}_{+\infty}$ are slow varying functions at infinity and $L^{j,k}_{0+}$ are slow varying functions close to 0+.

As a result, each element j,k of the spectral density matrix of a multi-variate LRD time series can be written as $f_{X,j,k} \sim g_{j,k}\lambda^{-(d_j+d_k)}$ for low frequencies λ. Similar to the mono-variate case, we can use a log-periodogram regression in low frequencies as a way to estimate d.

3.3 LRD and Mutual Information

Assuming (X) is Gaussian, a relation between the rate of decay of the auto-correlation of (X) and that of the Mutual Information $I(X_t, X_{t+h})$ can be established [14, 29]. One can easily prove that the mutual information of two multi-variate Gaussian random variables U, V is

$$I(U;V) = \frac{1}{2}\log\left(\frac{\det(\sigma_U)\det(\sigma_V)}{\det(\sigma)}\right)$$

where σ_U and σ_V are the corresponding covariance matrices and σ is the $2p \times 2p$ covariance matrix. Given that (X) is second-order stationary and Gaussian, one can show that [20, 21]

$$\begin{aligned} &I(X_t, X_{t+h}) - \log\det(\gamma_X(0)) \\ &= -\frac{1}{2}\left(\log\det(\gamma_X(0)\gamma_X(0) - \gamma_X(h)\gamma_X(h))\right) \\ &\sim \sum_{i=1}^{p} L^i_{+\infty}(h)h^{2(2d_i-1)} \end{aligned}$$

in the simple case where γ_X is diagonal. Here $(L^i_{+\infty})_{i=1\ldots p}$ are slow varying functions at infinity. Therefore one can assume a characteristic power-law decay of the mutual information based on the values of d, which relates our spectral method for characterizing sequential memory to the mutual information based approach in [26]. In particular, in the mono-variate case, that is $p = 1$, we have

$$\log I(X_t, X_{t+h}) \propto 2(2d-1)\log h.$$

That is, the slope of decay of mutual information w.r.t. separation in the log-log space corresponds to the LRD coefficient d. Unfortunately in the multivariate case, the slope does not give access, even with the Gaussian diagonal assumption, to individual estimates of the components of d.

3.4 Implementation at scale

Let us first focus on a detailed algorithmic presentation of the estimation procedure we designed to estimate LRD coefficients in long sequences of symbols.

Algorithm 1 details the actual implementation for a given sequence and Figure 2 presents it schematically. In order to scale the method to data sets comprising millions of sequences we run the procedure in a mini-batched manner, computing FFTs and OLSs in parallel, while the estimates for the coefficients d update a global estimate with a chosen learning rate. More details on the implementation are given in appendix.

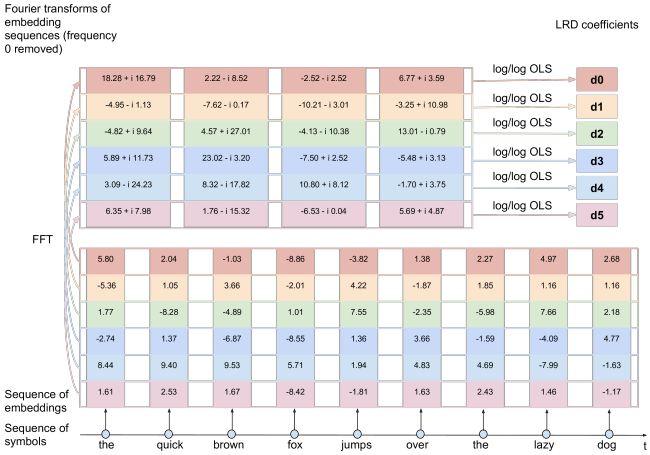

Figure 2: Schema of the log-periodogram estimation procedure employed in our study

Algorithm 1 Estimate LRD coefficients for a sequence of symbols

Require: L {padding length}, p {embedding dimension}, **E** {symbol embeddings}
Ensure: $d \in \mathbb{R}^p$
  embeddingSequence ← lookupEmbedding(**E**, symbolSequence)
  paddedEmbeddingSequence ← pad(embeddingSequence, L, **0**) {pad the beginning of sequence with zero valued vectors to obtain a sequence of length L}
  spectrum ← $|\text{RFFT}(\text{paddedEmbeddingSequence})[1:]|^2$ {remove the frequency 0 term}
  for $i \leftarrow 0$ to $p-1$ **do**
    d[i] ← OLS(log(range(1, L // 2 + 1), log(spectrum[:, i]))
  end for
  return d

3.5 Observations of LRD on actual sequential data sets

We now apply the estimation of the memory coefficient vector d to sequences of learned item embeddings in a language and a user-behavior dataset. It is worth pointing out that the log-periodogram estimate of LRD assumes that the time series are second-order stationary and the spectrum measurement using FFT uncovers only linear sequential dependency patterns. Although neither assumptions are guaranteed in any arbitrary time series, our method is sufficient to detect linear second order stationary LRD patterns if they exist, without guarantee that it will unravel any kind of non-linear or non-stationary LRD. Our empirical results show that such a linear LRD does exist in the sequences of word embeddings and item embeddings corresponding respectively to text documents and user/item interactions on YouTube.

3.5.1 LRD in word sequences. We start with measuring the LRD coefficients d on a subset of the Wikipedia dump consisting of concatenated Wikipedia articles (100 MB from Wikipedia) which keeps the sequential structure of the documents intact — no processing is done besides removing punctuation, converting all letters

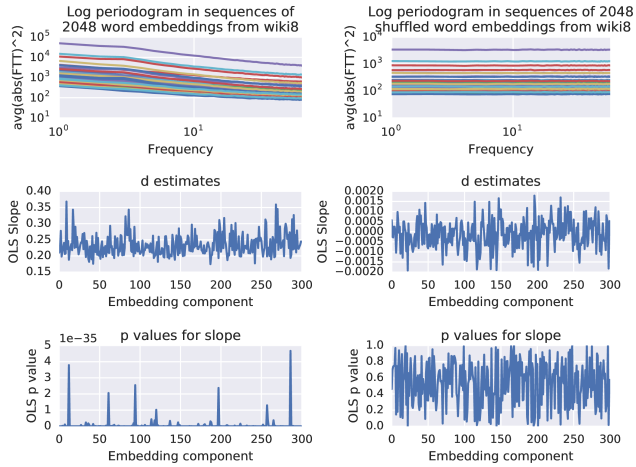

Figure 3: Spectral density estimate (in log space), the LRD coefficients and the p-value for the OLS regression on the Wikipedia dataset (left). Similar estimation (right) with sequences whose words have been randomly shuffled.

to lowercase and removing other artifacts. We break the documents into long sequences of 2048 words. Each sequence is then transformed into a multi-variate real-valued time series by mapping each word of the sequence to a pre-trained 300-dimensional Glove embedding [37], learned on the 2014 Wikipedia dump. Next, we compute the average of the squared magnitude $\widehat{f_X(\lambda)}$ of the Fourier Transform of the sequence of word embeddings. The slope of $\log\left(\widehat{f_X(\lambda)}\right)$ versus $\log(\lambda)$ on low frequencies is then estimated through OLS by minimizing the corresponding log-periodogram loss as shown in Equation 2. We tried different padding strategies for words whose embedding was unknown — skipping, zero padding and mean learned embedding padding — and did not find any substantial change in the resulting estimates of d.

We present the spectral density estimate, the LRD coefficients and the p-value for the OLS regression in Figure 3 (left). As shown in the figure, the spectral density estimates — each curve corresponding to FFT of one dimension of the word embedding — decay linearly near 0 in the log-log space, and the estimates for the coefficients d, with all the 300 dimensions shown in the x-axis, are significantly positive. These observations suggest that the time series under consideration is long range dependent according to the canonical statistical definition of LRD, which in return indicates that the input sequence is LRD. A higher coefficient demonstrates that a higher amount of memory is present. To give a confidence of our estimate, we also include the p-value of OLS estimating a slope of zero, that is $d = 0$. Extremely small p-values are returned, which again corroborates that the sequences are indeed LRD. Ideally, one would tailor the p-value tests to the particular setting of OLS in log-scale (which violates some assumptions on the distribution of errors), but this is outside the score of the paper. Detailed theory on the log-periodogram estimator can be found in [41].

We now proceed with a sanity check for the estimator of LRD through the log-periodogram method operating on vector-valued sequences of embedded symbols. On Figure 3 (right) we include the estimates on the same data-sets with the words randomly shuffled within sequences of 2048 words. Random shuffling dissolves any dependence patterns existing in the original data-set and gives a white-noise-like statistical structure to the embedding sequences. We can see that the spectral density no longer concentrate its mass around 0 (low frequencies) and our log-periodogram estimator gives close to 0 estimates for the LRD coefficient of the shuffled sequences with correspondingly high p-values. The side-by-side comparison showcased that our method is able to detect the presence of LRD (as in the original word sequences) vs not (as in the shuffled sequences).

3.5.2 LRD in user behavior sequences. Next we measure the LRD coefficients d on user behavior data. The data set consists of user generated interaction sequences in a large-scale anonymized dataset from YouTube to which we have access through employment at YouTube working on improving YouTube for users. Each sequence records a series of timestamped item ids corresponding to a given user u starting to access an item v at time t_i: $\left\{\xi^u_{t_i}\right\}_{i=1\ldots N_u}$ — where N_u is the number of interactions available for user u. We clip the sequences to at most 500 observations per user. This production data-set comprises of more than 200 million training sequences, more than 1 million test sequences and has an average sequence length of 200.

Different from the word sequences where each X_t involves a single symbol, $\xi^u_{t_i}$ includes multiple sub-symbols, each corresponding to one different aspect of the interaction: the item watched (from a vocabulary of 2 million items), the creator/publisher of the watched item (from a vocabulary of 1 million), the page the item was displayed (order of tens) and the OS employed by the user (orders of hundreds). The discrete values of these four groups of symbols are embedded with 128, 128, 32 and 32 dimensional real-valued vectors respectively. The embeddings are concatenated and trained as part of the sequential neural model aiming at predicting the next item the user will consume, which we are going to detail in section 5.2. With the learned embeddings, we follow the same procedure as described in the word sequence case to estimate the LRD coefficients. Figure 4 plots the spectral density of the embedding sequences. Again, the power law decay (left) and the linear decay in the log-log space (right) near 0 are the clear marks of a LRD pattern. Figure 1 (left) shows the estimated four groups of coefficients d. We notice that the embedding representing creators/publishers estimates larger coefficients d than the embedding representing individual items, indicating more LRD. Also, the software interface embedding carries the highest amount of LRD which is expected as it is less likely to change within short sequences of interactions. The maximum OLS p-value for the linear slopes in log-log scale being zero is 1.32×10^{-108}.

Another aspect in which these user-behavior sequences differ from the word sequences is the irregularly-spaced events. In this first work we do not take into account of that in order to use the Fast Fourier Transform algorithm readily when computing the spectral density of $(\widetilde{\xi_t})$. The Fourier transform however is well defined for irregularly spaced time stamps [5, 7, 40] and in future work we plan to apply dedicated estimators to improve our estimation under these settings.

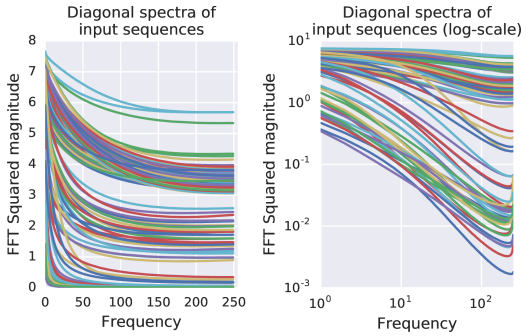

Figure 4: Spectral density of time series of embedded symbols in the original (left) and log-log space (right) where a linear decay is clear for several estimated spectra near 0.

4 LRD AND NEURAL ARCHITECTURE DESIGN

The estimation results above clearly indicate that there are long range dependence in sequences of words as well as sequences of user behavior data. The presence of LRD implies a power law decay of auto-covariance between present and past observation. In other words, as opposed to short memory sequences [38] where the auto-covariance decays at an exponential rate, past information has a long lasting footprint. Having confirmed the slow rate of decay of relevance of past observations, we now change the architectural design of standard RNNs accordingly.

Computational trade-offs and LRD. The presence of LRD in input sequences poses a major computational challenge for sequential modeling in live production systems. For tasks such as language understanding for auto-completion or behavioral prediction for recommendations, low latency deadlines (typically of the order of 100ms at most) have to be met by models. Such a tight constraint means that model architects need to find efficient ways to spend computational budgets in order to take information lying far in the past into account without incurring too much delay. The power-law decay in LRD sequences does imply a slower than expected decrease of the relevance of past information. It, however, also suggests that the most recent inputs are of higher relevance.

EvolutiveRNNs (EvoRNNs) for long LRD sequences. As information located far into the past is relevant but with a lower signal to noise ratio than the more recent observations, it is thus sub-optimal to spend the computational budget equally across inputs of different time steps as in standard RNNs. Our insight is to spend our add/multiply budget at serving time in proportion of the amount of correlation between successive inputs and the final output.

We therefore propose an evolutive recurrent neural network architecture, named EvoRNN for LRD sequences. EvoRNN spends most of its computational budget on the most recent inputs while lowering the cost of considering inputs located further into the past. As looking further back into the past is less expensive for such models, practitioners can also consider longer sequences under the same latency constraints.

The main difference between the standard RNN equation 1 and the architecture we propose is that the size of the RNN cell being called depends on the distance to the end of the sequence:

$$[\hat{Y}_t, M_t]^T = \Phi_{L-t,\theta_{L-t}}(X_t, P_{L-t}M_{t-1}) \quad (3)$$

where L is the number of symbols in the sequence we consider, t the current position and P_{L-t} a rectangular matrix (whose parameters are learned) which re-sizes the previous state M_{t-1} so that it has the dimension excepted for the state of cell $\Phi_{L-t,\theta_{L-t}}$. In our experiments if the state dimensions are already compatible between the previous cell and the next cell we remove the projection matrix P_{L-t}. We typically have few of these projections and plan to speed them up using fast structured linear operators such as randomized projections based on FFTs [54].

We consider two decay schemes for the number of units of the RNN cells as the distance to the end of the sequence – where the prediction is made at serving time – increases: a power-law decay (PowerLawEvoRNN) and an exponential decay (ExpEvoRNN). Figure 5 illustrates our proposed architecture if a power-law decay of the computational intensity is chosen. The computational budget

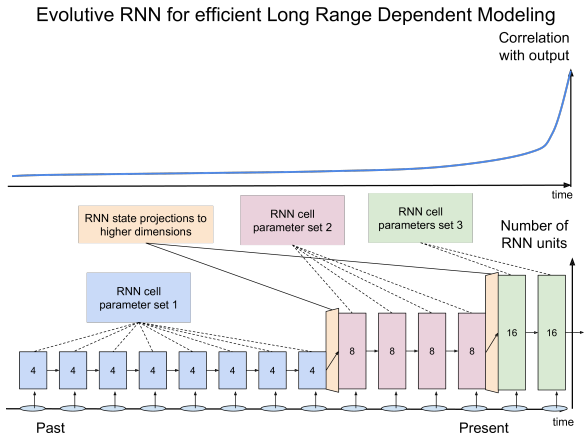

Figure 5: PowerLawEvoRNN architecture: the power-law decay of the computational intensity per input mirrors the power-law decay of correlation of inputs with the final output of the sequence in the presence of LRD.

of an RNN cell with h hidden units scales as $O(h^2)$ to incorporate one input. The resulting distribution of compute time is presented in Figure 6. One can appreciate how EvoRNN architectures rely on much fewer add/multiplies to process long sequences.

5 EXPERIMENTAL RESULTS WITH EVORNN

We now compare the performance of EvoRNN architectures and standard RNNs on a benchmark language modeling task and a sequential recommendation task. We show that on both tasks EvoRNNs outperform or reach comparable performance with the more computationally expensive baseline.

5.1 LM1B language modeling task

The Billion word data set [9] is a standard benchmark for language modeling [3, 10, 32] aimed at predicting the next word in a text. We slightly modify the benchmark to create a long range prediction task involving longer sequences. Sequences of 128 words are considered and the model's task is now to predict the last 4 words.

Model	Sub-sequence lengths	# of hidden units	# of add/mul.
LM baseline	128	2048	536870912
LM PowerLawEvoRNN	64, 32, 16, 8, 4, 4	64, 128, 256, 512, 1024, 2048	24903680
LM ExpEvoRNN	108, 4, 4, 4, 4, 4	64, 128, 256, 512, 1024, 2048	22790144
Seq. rec. baseline	512	256	33554432
Seq. rec. PowerLawEvoRNN	256, 128, 64, 32, 32	32, 64, 128, 256, 256	6029312
Seq. rec. ExpEvoRNN	384, 32, 32, 32, 32	34, 69, 138, 276, 276	6080928
Seq. rec. ExtrExpEvoRNN	480, 8, 8, 8, 8	2, 8, 64, 256, 1024	8948096

Table 1: RNN architectures employed in the sequential recommendation and language modeling tasks. Although they learn more parameters, EvoRNNs require much less compute time than baselines and therefore can be served under lower latency constraints. Here, # add/mutiply give asymptotic complexity estimates to consider in relative magnitude.

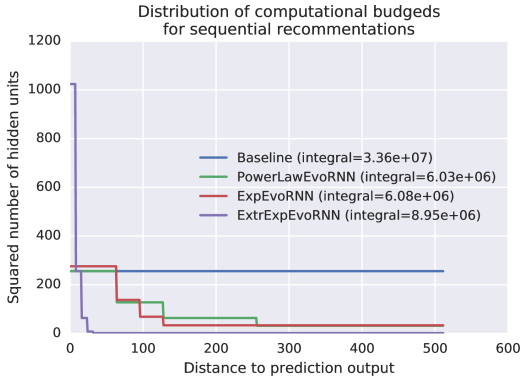

Figure 6: Computational budget – $O(h^2)$ where h is the number of hidden units – of a baseline RNN and its PowerLawEvoRNN and ExpEvoRNN counterparts as a function of distance to end of sequence.

The baseline model is a LSTM [23] following the implementation of [25]. EvoRNNs follow the same setup except the different distribution of compute resources. We consider two variants of the EvoRNN architecture: a power law decay variant and an exponential decay variant. The cell sizes and computational footprint of both variants are detailed in Table 1. As an example, the power law decay variant break the input sequence into six subsquences of length 64, 32, 16, 8, 4 and 4, each using RNNs of hidden units of 64, 128, 256, 512, 1024 and 2048, with most compute spending near the end of the sequence. [1] The total number of add/multiply performed using the baseline model as well as EvoRNNs with different scheduling are shown in the last column.

Figure 7 shows that on this language modeling task both variants of the cheaper EvoRNN architectures out-perform the baseline model with only fraction of compute resources, as indicated by the wall time (bottom two plots) in addition to the estimate in Table 1. This can be attributed to the implicit architectural prior of EvoRNN giving fewer degrees of freedom to parameters involved in processing inputs located further into the past, which inherently

[1] Note that as RNN cells of different number of hidden units are instantiated through the sequence, additional projection matrices, one between two sub-sequences, are learned in EvoRNNs. We can further save the computational cost by using fast random projections as in [54].

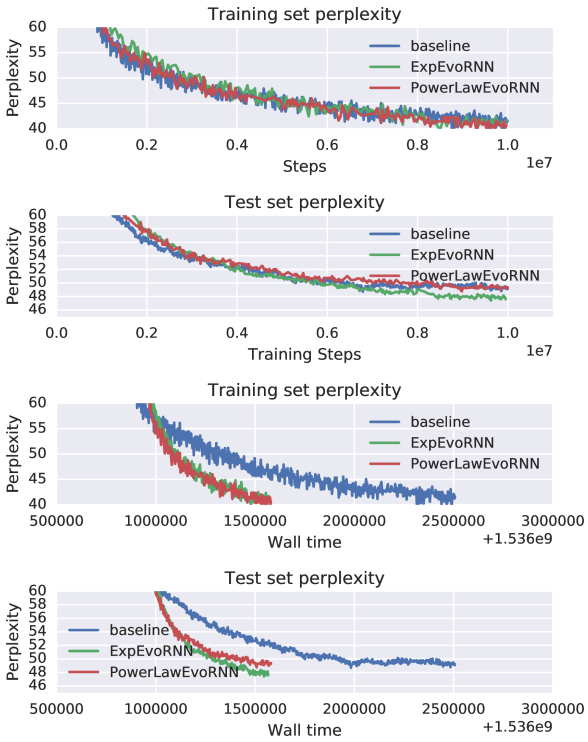

Figure 7: Performance results on language modeling task. EvoRNN architectures provide better performance than the baseline model and train considerably faster. The increase in training speed is expected as the EvoRNN architectures spend less compute-time on earlier inputs.

has less signal for predicting the next word near the end of the sequence. Architectures having fewer hidden units for these inputs may resist more robustly to the lower levels of signal to noise ratio present at the beginning of the sequence.

5.2 Sequential recommendation task

Next we consider a sequential recommendation task, where the sequential recommender under consideration serves users accessing a browsing page on which impressions are displayed. It has access

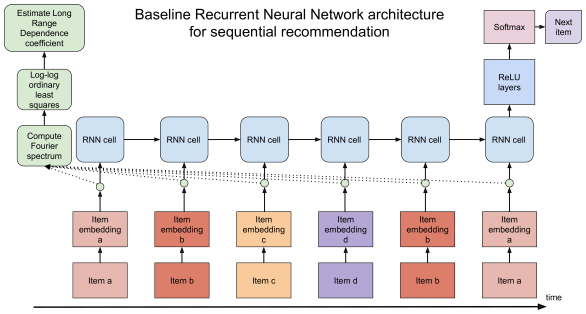

Figure 8: Architecture for the sequential recommender. Input items are embedded into dense real-valued embeddings as described in section 3.5.2 and sent through an RNN to predict the next item to be consumed. Left part of the figure, shown in green blocks, depicts the estimation process we detailed in section 3.5.1

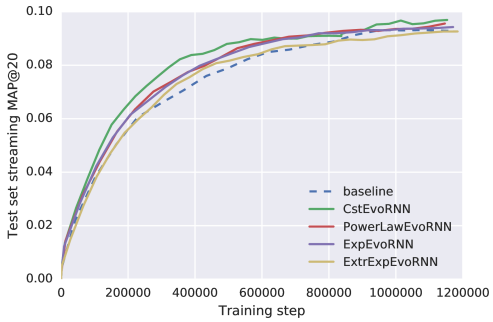

Figure 9: Performance results on the sequential recommendation task. For a lower computational budget, the EvoRNNs provide similar performance to the more expensive constant size baseline RNN. Although they use more memory because more parameters are learned in the present implementation, the PowerLaw and Exp EvoRNN networks need much fewer add/multiplies to compute a prediction as they spend less time processing inputs located further in the past. The extremely myopic ExtrExp RNN however does not perform as well as the baseline although it has many more free parameters and a higher computational budget than the other EvoRNNs. CstEvoRNN is an EvoRNN with as many independently learned cells as PowerLawEvoRNN but all as large as the baseline's cell. In spite of the increased number of free parameters, CstEvoRNN does not show much improvement.

to historical interactions, *i.e.,* watched items, from the same user identified by the same personal account. This sequential neural model nominating items for recommendation therefore maps the sequence of observations $\left\{\begin{matrix}\xi u\\ \xi t_i\end{matrix}\right\}_{i=1\ldots N_u}$ to a predicted item ξ.

Neural recommender systems attempt at foreseeing the interest of users under extreme constraints of latency and scale. We define the task as predicting the next item the user will consume given a recorded history of items already consumed. Such a problem setting is indeed common in collaborative filtering [27, 43] recommendations. While the user history can span over months, only watches from the last 7 days are used for labels in training and watches in the last 2 days are used for testing. The train/test split is 90/10%. The test set does not overlap with the train set and corresponds to the last temporal slice of the dataset.

Figure 8 shows the underlying architecture powering the recommender. It relies on an RNN, a gated recurrent unit (GRU) [13], to read through the sequence of past observations and predict the ID of the next item to be consumed by the user. The network parameters are trained using Adagrad to minimize a weighted cross-entropy loss. The embedding dimensions and RNN cells are summarized in appendix (Table 2). The model is a standard GRU fed with embedded symbols (after concatenation) producing predictions on the items the user will click with a softmax layer trained by negative sampling.

Table 1 shows the number of hidden units used in our baseline and the EvoRNNs for this task as well. Similarly, EvoRNNs only need a fraction of the add/multiplies used in the baseline RNN.

We report the Mean-average-precision-at-20 (MAP@20) [19] as the main performance metric. Figure 9 shows the progress of the MAP@20 score for the baseline and different variants of the EvoRNNs, up to 1.2 million steps of training updates. We can see that 1) Variants of EvoRNNs reaches comparable performance as the baseline model, which uses 5 times more compute than EvoRNNs; 2) ExtrExpRNN, a myopic RNN designed to spend all of its compute near the very end of the sequences, and little on inputs in the past performed slightly worse, which confirms that the user behavior data is indeed LRD, and completely ignoring inputs from the past results in performance degradation.

5.3 Conclusion and discussion on experiments

After having demonstrated that the sequences of inputs we consider are LRD, we also showed that a parsimonious use of computational spending is sufficient to produce competitive expressive models for language modeling and sequential recommendation tasks. The models we introduce, EvoRNNs, dedicate their serving time computational budget in priority to recent inputs while also leveraging information located further into the past. While the new architectures train more parameters (learning multiple RNN cells) in their present implementation, they help process longer sequences of inputs under a given latency deadline which is crucial to improve predictive accuracy for LRD sequential prediction tasks. In future work we aim at reducing the memory footprint by reusing parameters when processing earlier inputs in the sequence as larger parameter matrices can be projected to fewer dimensions.

6 CONCLUSION

While the issue of LRD has been considered widely in neural sequential modeling, LRD has not been quantified methodically for the corresponding sequences of inputs. For tasks such as language understanding and sequential recommendations — where neural sequential models are pervasive and provide state-of-the-art performance — model dependent gradient based considerations dominate when it comes to measuring LRD. In the present work, we employed a well established LRD theory for real-valued time series on sequences of vector-valued item embeddings to estimate LRD in sequences of discrete symbols belonging to large vocabularies. The resulting estimates of LRD coefficients unraveled new exploratory insights on modeling sequences of words and user interactions. Considering the power law decay of relevance of past inputs led to the construction of new recurrent architectures: EvoRNNs. EvoRNNs showed a performance at worst comparable with state-of-the-art baselines for language modeling and sequential recommendations using only a fraction of the computational cost.

REFERENCES

[1] Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2016. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*. 1120–1128.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[4] Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. 2018. Factorized Recurrent Neural Architectures for Longer Range Dependence. In *International Conference on Artificial Intelligence and Statistics*. 1522–1530.

[5] Francois Belletti, Evan Sparks, Alexandre Bayen, and Joseph Gonzalez. 2017. Random projection design for scalable implicit smoothing of randomly observed stochastic processes. In *Artificial Intelligence and Statistics*. 700–708.

[6] Jan Beran. 2017. *Statistics for long-memory processes*. Routledge.

[7] David R Brillinger. 1981. *Time series: data analysis and theory*. Vol. 36. Siam.

[8] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*. 77–87.

[9] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005* (2013).

[10] Minmin Chen, Jeffrey Pennington, and Samuel S Schoenholz. 2018. Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal Propagation in Recurrent Neural Networks. *arXiv preprint arXiv:1806.05394* (2018).

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[12] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704* (2016).

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[14] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

[15] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[16] Robin Devooght and Hugues Bersini. 2017. Long and short-term recommendations with recurrent neural networks. In *Conference on User Modeling, Adaptation and Personalization*. ACM, 13–21.

[17] Paul Doukhan, George Oppenheim, and Murad Taqqu. 2002. *Theory and applications of long-range dependence*. Springer Science & Business Media.

[18] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *ICASSP*. IEEE, 6645–6649.

[19] Matthieu Guillaumin, Thomas Mensink, Jakob Verbeek, and Cordelia Schmid. 2009. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 309–316.

[20] Dongning Guo, Shlomo Shamai, and Sergio Verdú. 2005. Additive non-Gaussian noise channels: Mutual information and conditional mean estimation. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*. IEEE, 719–723.

[21] Dongning Guo, Shlomo Shamai, and Sergio Verdú. 2005. Mutual information and minimum mean-square error in Gaussian channels. *IEEE Transactions on Information Theory* 51, 4 (2005), 1261–1282.

[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[23] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[24] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. 2016. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. *arXiv preprint arXiv:1612.05231* (2016).

[25] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* (2016).

[26] Henry W Lin and Max Tegmark. 2016. Criticality in formal languages and statistical physics. *arXiv preprint arXiv:1606.06737* (2016).

[27] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[28] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[29] David JC MacKay. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[30] Benoıt B Mandelbrot and Ian Stewart. 1998. Fractals and scaling in finance. *Nature* 391, 6669 (1998), 758–758.

[31] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182* (2017).

[32] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[34] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 746–751.

[35] John Miller and Moritz Hardt. 2018. When Recurrent Models Don't Need To Be Recurrent. *arXiv preprint arXiv:1805.10369* (2018).

[36] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[37] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[38] Vladas Pipiras and Murad S Taqqu. 2017. *Long-range dependence and self-similarity*. Vol. 45. Cambridge university press.

[39] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 130–137.

[40] Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in neural information processing systems*. 1177–1184.

[41] Peter M Robinson. 1995. Log-periodogram regression of time series with long range dependence. *The annals of Statistics* (1995), 1048–1072.

[42] Gennady Samorodnitsky et al. 2007. Long range dependence. *Foundations and Trends® in Stochastic Systems* 1, 3 (2007), 163–257.

[43] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. ACM, 285–295.

[44] Elena Smirnova and Flavian Vasile. 2017. Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*. ACM, 2–9.

[45] Didier Sornette. 2006. *Critical phenomena in natural sciences: chaos, fractals, selforganization and disorder: concepts and tools*. Springer Science & Business Media.

[46] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[47] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H Chi. 2019. Towards Neural Mixture Recommender for Long Range Dependent User Sequences. *arXiv preprint arXiv:1902.08588* (2019).

[48] Trieu H Trinh, Andrew M Dai, Thang Luong, and Quoc V Le. 2018. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144* (2018).

[49] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio.. In *SSW*. 125.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[51] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. 2017. On orthogonality and learning recurrent networks with long term dependencies. *arXiv preprint arXiv:1702.00071* (2017).

[52] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. ACM, 495–503.

[53] Doris Xin, Nicolas Mayoraz, Hubert Pham, Karthik Lakshmanan, and John R Anderson. 2017. Folding: Why Good Models Sometimes Make Spurious Recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 201–209.

[54] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. 2015. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*. 1476–1483.

[55] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

APPENDIX

We now produce the pseudo code and instructions to help reproduce our experiments.

Implementation of estimators for memory coefficients $d \in \mathbb{R}^p$ on large data sets

We only described in Algorithm 1 how we implemented the log-periodogram method to estimate LRD coefficients for a single sequence of symbols. We now give more details on how the estimate at the scale of an entire data set comprising millions of such sequences is computed. We employ an exponential moving average based estimation as follows. Consider embeddings of p dimensions, the LRD coefficient vector we estimate is therefore in $\mathbb{R}^d$. We start with a first estimate $\widehat{d_0} \in \mathbb{R}^p$ and chose a learning rate sequence $(\alpha_i)_{i\in\mathbb{N}}$. Sequences are grouped in mini-batches (typically of 256 sequences) on which log-periodogram estimates are computed independently (in parallel) by running Algorithm 1. Considering a batch size B and the step i of the SGD, let $(\widetilde{d}_{i,j})_{j=0,\ldots,B-1}$ the resulting batch of LRD coefficient estimates in $\mathbb{R}^p$. The previous estimate for the overall LRD coefficient is updated as follows:

$$\widehat{d}_{i+1} = (1-\alpha_i)\widehat{d}_i + \alpha_i \frac{1}{B}\sum_{j=0}^{B-1} \widetilde{d}_{i,j}.$$

In our experiments the learning rate follows a standard inverse decay scheme ($\sim \frac{1}{i}$).

Implementation of Neural architectures

The appendix now presents the general neural architecture for symbol prediction considered in the paper before describing the particular implementations for language understanding and behavior prediction.

Generic RNN for symbol prediction. We now describe the generic architecture employed in the paper to predict the next symbol in a sequence of symbols. Let Vocab $= \{0 \ldots V-1\}$ be the vocabulary of possible symbol values. After having chosen an input embedding size d^{input} we create a table of input embeddings:

$$E^{\text{input}} = (e_i^{\text{input}})_{i=1\ldots V-1}$$

where $\forall i \in \{1 \ldots V-1\}, e_i^{\text{input}} \in \mathbb{R}^{d^{\text{input}}}$.

Consider a sequence of input symbols $(s_0, \ldots, s_{L-1})$. We attempt to predict the next symbol s_L, i.e. the label. The very first processing step transforms the sequence of L symbols in Vocab into a sequence of real valued vectors:

$$(s_0, \ldots, s_{L-1}) \rightarrow (e_{s_0}^{\text{input}}, \ldots, e_{s_{L-1}}^{\text{input}}).$$

Now, $(e_{s_0}^{\text{input}}, \ldots, e_{s_{L-1}}^{\text{input}})$ constitutes the sequence that serves as input into the first (and possibly only) RNN layer: RNN_0. Let us describe how the i^{th} RNN layer processes its inputs denoted by $(x_0^i, \ldots, x_{L-1}^i)$.

Consider an RNN layer taking inputs in $\mathbb{R}^{d^i}$ and outputting an RNN state in $\mathbb{R}^{d^{i+1}}$ as well as an output in $\mathbb{R}^{d^{i+1}}$:

$$(x_0^i, \ldots, x_{L-1}^i) \rightarrow (x_0^{i+1}, \ldots, x_{L-1}^{i+1})$$

where the outputs $(x_0^{i+1}, \ldots, o_{L-1}^{i+1})$ and the states $(h_0^i, \ldots, h_{L-1}^i)$ are generated — given an initial value $\text{init}^i \in \mathbb{R}^{d^{i+1}}$ for the RNN state which can be set to 0, randomized or learned as a parameter — by the recurrent relation:

$$h_0^i = \text{init}^i, \tag{4}$$

$$(x_{t+1}^{i+1}, h_{t+1}^i) = \text{RNNCell}_{\theta^i}^i(x_t^i, h_t^i). \tag{5}$$

Typically, $\text{RNNCell}_\theta^i(x_t^i, h_t^i)$ (if RNNCell is an Elman/Jordan cell, an LSTM cell, a GRU cell for instance) involves $O(d^i \times d^{i+1})$ storage for learned parameters and $O(d^i \times d^{i+1})$ add/multiplies because of the underlying matrix/vector multiplications.

For an RNN network of depth D, i.e. with D stacked RNN layers $\text{RNN}^0, \ldots, \text{RNN}^{D-1}$, the final output of interest is

$$(x_L^D).$$

One may also predict further symbols such as $s_{L+1}, s_{L+2}, \ldots, s_{L+H}$. This is readily possible by unfolding the RNN recurrence further in time after the time steps t. In that case, the RNN outputs of interest are

$$(x_L^D, x_{L+1}^D, \ldots, x_{L+H}^D).$$

In either case, a decoding layer (which may in fact comprise several Fully Connected layers) transforms the RNN outputs into the final outputs used for symbol prediction. Let $\text{Dec}_{\theta^{\text{Dec}}}$ be the final layer taking inputs in $\mathbb{R}^{d^D}$ and with outputs in $\mathbb{R}^{d^O}$. The decoded output sequence is obtained as follows:

$$(x_L^D, x_{L+1}^D, \ldots, x_{L+H}^D) \rightarrow (y_L, y_{L+1}, \ldots, y_{L+H})$$

where

$$(y_L, y_{L+1}, \ldots, y_{L+H}) = (\text{Dec}_{\theta^{\text{Dec}}}(x_L^D), \ldots, \text{Dec}_{\theta^{\text{Dec}}}(x_{L+H}^D)).$$

Finally, to match the vector valued outputs with the labels, we classically employ an output side embedding table E^{output} (which may be in fact the input side embedding table E^{input} itself if the corresponding embedding dimensions are equal):

$$E^{\text{output}} = (e_i^{\text{output}})_{i=1\ldots V-1}$$

where $\forall i \in \{1 \ldots V-1\}, e_i^{\text{output}} \in \mathbb{R}^{d^{\text{output}}}$.

The output embedding table encodes the sequence of symbols that have to be predicted, i.e. the sequence of labels, as follows:

$$(s_L, \ldots, s_{L+H}) \rightarrow (e_{s_L}^{\text{output}}, \ldots, e_{s_{L+H}}^{\text{output}}).$$

Similarity scores between the label embeddings and the decoded outputs is computed with a softmax layer using the inner product as a similarity metric:

$$\text{score} = \sum_{j=L}^{L+H} exp\left(\left\langle y_j, e_{s_j}^{\text{output}} \right\rangle\right) - \log\left(\sum_{k=0\ldots V-1} exp\left(\left\langle y_j, e_k^{\text{output}} \right\rangle\right)\right)$$

The score corresponds to the log probability of the data prescribed by the parametric RNN model. Classically, we use a Stochastic Gradient Descent algorithm iterating on an entire data set of sequences to maximize the score w.r.t. to all the paremeters values. Here the embedding table parameters as well as the RNN and decoding layer are trained. To accelerate the computation of the softmax during training we employed the standard technique of sampling negatives (see for instance sampled_softmax_loss in Tensorflow).

Generic EvoRNN architecture for symbol prediction. The EvoRNN architecture we designed only modifies the recurrent equation 5. In all the following, t denotes the position of a symbol w.r.t. to the *beginning* of the input sequence while t' denotes the position w.r.t. to the *end* of the input sequence, i.e.

$$t' = L - 1 - t.$$

The first step when instantiating an EvoRNN is to create a map from the distance to the end of the input sequence to an RNNCell:

$$\text{RNNCellMap} : t, L \longrightarrow \text{RNNCellArray}[t' = L - 1 - t] \in \text{RNNCells}$$

which maps a sequential position (counted from the end of the input sequence) to an RNNCell in a set of chosen RNNCells. RNNCellArray is an array of pointers to RNNCells. Typically, a template cell will be chosen such as GRU and the cells in RNNCells will only differ by their number of outputs.

Let us consider a practical example with GRU as the template cell. We instantiate 4 different RNNCells with output dimensions 2, 4, 8, 16: $\text{GRU}_2, \text{GRU}_4, \text{GRU}_8, \text{GRU}_{16}$. To create a "power-law" decay in the size of the cell w.r.t. the distance to the end of the sequence, we specify RNNCellMap as follows:

$$\begin{aligned}
&\text{if } 0 <= L-1-t < 2, && \mathit{RNNCellMap}(t, L) = \text{GRU}_{16}\\
&\text{if } 2 <= L-1-t < 6, && \mathit{RNNCellMap}(t, L) = \text{GRU}_{8}\\
&\text{if } 6 <= L-1-t < 14, && \mathit{RNNCellMap}(t, L) = \text{GRU}_{4}\\
&\text{if } 14 <= L-1-t, && \mathit{RNNCellMap}(t, L) = \text{GRU}_{2}.
\end{aligned}$$

Once an RNNCellMap has been specified, the RNN recurrent 5 can be modified naively as follows:

$$h_0^i = \text{init}^i, \tag{6}$$

$$(x_{t+1}^{i+1}, h_{t+1}^i) = \text{RNNCellMap}^i(t, L)(x_t^i, h_t^i). \tag{7}$$

Now, as the RNN state dimensions differ between GRU_2 and GRU_4 (being equal to 2 and 4 respectively), the naive EvoRNN equation 9 is flawed and ill-defined. To make the RNN state dimensions match, we create a projection map (ProjMap)

$$\text{ProjMap} : t \in \{0 \dots L-1\} \longrightarrow \text{ProjArray}[t] \in \text{ProjSet}$$

that maps a given distance to the end of the input sequence to a projection matrix as follows:

$$\text{if stateDim}\left(\text{RNNCellMap}(t-1, L)\right) \text{ != stateDim}\left(\text{RNNCellMap}(t, L)\right),$$

$$\text{then ProjMap}(t, L) = M_t$$

$$\text{else ProjMap}(t, L) = I$$

with

$$M_t \in \mathbb{R}^{\text{stateDim}(\text{RNNCellMap}(t,L)), \text{stateDim}(\text{RNNCellMap}(t-1,L))}$$

Again, ProjArray is an array of pointers to real valued matrices of various sizes.

Therefore, the well formed recurrent definition of the EvoRNN writes

$$h_0^i = \text{init}^i, \tag{8}$$

$$(x_{t+1}^{i+1}, h_{t+1}^i) = \text{RNNCellMap}^i(t, L)(x_t^i, \text{ProjMap}(t, L)(h_t^i)). \tag{9}$$

EvoRNN is therefore well defined as a standard RNNCell with the distinction that it needs to be informed, for each input, of its position relative to the end of the sequence. Such a property is very advantageous as it makes it trivial for EvoRNN to replace any RNN model and work with batched sequences of different lengths.

Parameter	Value
Input embedding dimensions	128 + 128 + 32 + 32
RNN cell size	256
Softmax training negative sampling	Learned unigram
Output embedding dimension	256

Table 2: Parameters used in the baseline neural recommender for user behavior. Only the RNN cell is modified and swapped with an EvoRNN. The embeddings given as inputs to the RNN consist of the concatenated embeddings for the video, creator/publisher, interface and page type.

Implementation of experiments

We implemented all our experiments in Tensorflow.

Implementation of RNN for language prediction experiment. The baseline implementation corresponds to [25] and can be found at github.com/tensorflow/models/tree/master/research/lm_1b.

The only architectural modification we made was to the RNN code. We also slightly changed the training strategy in that only 4 labels after the end of a 128 symbol sequence would be used to train and test the model's predictive ability. The input and output vocabulary sizes are 800K, 512 embedding dimensions are used, the number of units in the LSTM is 2048 and there is only one layer. The decoder is a dense matrix multiplies projecting down to 512 dimensions (the output embedding dimension). 8192 negative samples are employed during training in the softmax layer. We employ standard Stochastic Gradient descent with a learning rate of 0.2 and gradient clipping.

Implementation of RNN in behavior prediction experiment. Each input symbol corresponds to 4 concatenated sub-symbols corresponding respectively to an item embedding, a creator/publisher embedding, a home vs publisher page embedding and finally an interface embedding. Each input sub-symbol is endowed with its own embedding table. 128 dimensions are used to embed items ids, 128 to embed publishers ids, 32 to embed the browsed page type and 32 for the interface type. There is only one RNN layer comprising a GRU cell with a 256 dimensional output. The softmax is trained with 10000 negative samples per batch and the sampling distribution we use is a learned unigram distribution (the sampler starts sampling negatives uniformly at random and progressively learns to mimic the historical distribution of unigrams in the data). The output embedding table comprises 256 dimensions. We use a learning rate of 0.1 with Adagrad and gradient clipping. The hyper-parameters we use result of careful hyper-parameter tuning of the baseline. Table 2 gives a summarized view of the critical architectural parameters of the baseline we used on our proprietary data set.