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:

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 loses 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:

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

(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, 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
We follow the presentation of LRD given in [38] and consider a doubly to
2.1 LRD in real-valued time series
2 RELATED WORK
[38] delving in the topic of LRD, we recall the definition of slow varying is also well defined and verifies defined and do not change over time. The spectral density and its auto-covariance function \( \gamma(\lambda) \)
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 decreasing auto-covariance
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 latency constraints.

(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 \( \gamma_X(h) = \text{Cov}(X_t, X_{t+h}) \) are well defined and do not change over time. The spectral density \( f_X(\lambda) \) 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 if there exists a real \( d \in (0, \frac{1}{2}) \), referred to as the LRD coefficient of \( (X) \), such that

\[
\gamma_X(h) = L_\infty(h)h^{2d-1}, \quad \text{or equivalently } f_X(\lambda) = L_0(\lambda)\lambda^{-2d},
\]
where \( L_\infty \) 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 \( \frac{1}{N}\sum_{t=1}^{N}X_t \) decays much slower as \( L_\infty(N)N^{2d-1} \) where \( L_\infty \) 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 \( f_X(\lambda) = \mathcal{FFT}_N[\lambda] = \frac{1}{\sqrt{N}} \sum_{t=1}^{N} X_t e^{-it\lambda} \) 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

\[
\log \left( f_X(\lambda) \right) = a + b \log(\lambda)
\]
by ordinary least squares regression in the domain of low frequencies [41].

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 \( h^{2d-1} \). 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.
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 $10^3$ 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 $\gamma_X(h) = \text{Cov}(X_t, X_{t+h})$ 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 takes values in $\mathbb{R}^{p \times p}$. By definition $\gamma_X$ and $f_X$ satisfy $\forall j, k \in 1 \ldots p, \forall h \in \mathbb{Z}$,

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

**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 $(d)^{p \times 1} \in (0, \frac{1}{2})^p$ such that

$$
\gamma_X, j, k(h) = L^{d_{j+k}}_{+\infty}(h) h^{d_{j+k+1}}
$$

or equivalently

$$
f_X, j, k(\lambda) = L^{d_{j+k}}_{0\infty}(\lambda) \lambda^{-(d_{j+k}+d_k)}
$$
where \( L_{j,k}^0 \) are slow varying functions at infinity and \( L_{j,k}^1 \) 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 \( f_{X_i,j,k} \sim g_{j,k}\lambda^{-d_j+d_k} \) for low frequencies \( \lambda \). 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 \( I(X_i,X_{i+h}) \) 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 \( \Sigma_U \) and \( \Sigma_V \) are the corresponding covariance matrices and \( \Sigma \) is the \( 2p \times 2p \) covariance matrix. Given that \((X)\) is second-order stationary and Gaussian, one can show that [20, 21]

\[
I(X_i, X_{i+h}) \sim -\log \det (\gamma_X(0)) = -\frac{1}{2} \log \det (\gamma_X(h)\gamma_X(h)) = \sum_{i=0}^{p} L_{i,00}^i(h)h^{2(2d_i-1)}
\]

in the simple case where \( \gamma_X \) is diagonal. Here \( (L_{i,00}^i)_{i=1…p} \) 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

\[
\log I(X_i, X_{i+h}) \propto 2(2d_i - 1) \log h.
\]

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 the appendix.

#### 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
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 concentrates 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).

### 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 $t_i$:

$$\{s_{t_i}^u\}_{i=1, \ldots, N_u}$$

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, $s_{t_i}^u$ 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 $1.32 \times 10^{-108}$.

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 $(\mathbf{S}_t)$. 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.
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:

$$\hat{y}_t, M_{t} = \Phi_{L-t \cdot \theta_{L-t}}(X_t, P_{L-t}M_{t-1})$$

(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 $\Phi_{L-t \cdot \theta_{L-t}}$. 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 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.
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 sub-sequences 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 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 to historical interactions, i.e., watched items, from the same user

---

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].
identified by the same personal account. This sequential neural model nominating items for recommendation therefore maps the sequence of observations \( \{ y_{t_i} \}_{i=1 \ldots N_o} \) to a predicted item \( \xi \).

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 in 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.
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 $d_0 \in \mathbb{R}^P$ 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 $(d_{i,j})_{j=0,\ldots,B-1}$ the resulting batch of LRD coefficient estimates in $\mathbb{R}^P$. The previous estimate for the overall LRD coefficient is updated as follows:

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

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

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 $\text{Vocab} = \{0 \ldots V - 1\}$ be the vocabulary of possible symbol values. After having chosen an input embedding size $d^{\text{input}}$ we create a table of input embeddings:

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

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

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 $\text{Vocab}$ into a sequence of real valued vectors:

$$(s_0, \ldots, s_{L-1}) \rightarrow (e^{\text{input}}_0, \ldots, e^{\text{input}}_{L-1}).$$

Now, $(e^{\text{input}}_0, \ldots, e^{\text{input}}_{L-1})$ constitutes the sequence that serves as input into the first (and possibly only) RNN layer: $\text{RNN}_0$. Let us describe how the $i$-th RNN layer processes its inputs denoted by $(x^{(i)}_0, \ldots, x^{(i)}_{L-1})$.

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 $\mathbb{R}^{d^{(i+1)}}$:

$$(x^{(i)}_0, \ldots, x^{(i)}_{L-1}) \rightarrow (x^{(i+1)}_0, \ldots, x^{(i+1)}_{L-1})$$

where the outputs $(x^{(i+1)}_0, \ldots, x^{(i+1)}_{L-1})$ and the states $(h^{(i+1)}_0, \ldots, h^{(i+1)}_{L-1})$ are generated — given an initial value $\text{init}^i \in \mathbb{R}^{d^{(i)}}$ for the RNN state which can be set to 0, randomized or learned as a parameter — by the recurrent relation:

$$h^{(i)}_0 = \text{init}^i,$$

$$(x^{(i+1)}_0, h^{(i+1)}_0) = \text{RNNCell}^i(x^{(i)}_0, h^{(i)}_0).$$

Typically, $\text{RNNCell}^i(x^{(i)}_0, h^{(i)}_0)$ (if $\text{RNNCell}$ 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/multiplications 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

$$(x^{(D)}_L).$$

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

$$(x^{(D)}_L, x^{(D)}_{L+1}, \ldots, x^{(D)}_{L+H}).$$

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 $\mathbb{R}^{d^{(output)}}$. The decoded output sequence is obtained as follows:

$$(x^{(D)}_L, x^{(D)}_{L+1}, \ldots, x^{(D)}_{L+H}) \rightarrow (y_L, y_{L+1}, \ldots, y_{L+H})$$

where

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

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

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

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

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

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

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

$$\text{score} = \sum_{j=L}^{L+H} \exp \left( \langle y_j, e^{\text{output}}_j \rangle \right) - \log \left( \sum_{k=0}^{V-1} \exp \left( \langle y_k, e^{\text{output}}_k \rangle \right) \right)$$

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 parameters 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:

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

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: GRU$_2$, GRU$_4$, GRU$_8$, 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:

1. if $0 \leq L - 1 - t < 2$, \[ \text{RNNCellMap}(t, L) = \text{GRU}_{16} \]
2. if $2 \leq L - 1 - t < 6$, \[ \text{RNNCellMap}(t, L) = \text{GRU}_{8} \]
3. if $6 \leq L - 1 - t < 14$, \[ \text{RNNCellMap}(t, L) = \text{GRU}_{4} \]
4. if $14 \leq L - 1 - t$, \[ \text{RNNCellMap}(t, L) = \text{GRU}_{2} \]

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

$$
h_t^i = \text{init}^i, \quad (x_t^{i+1}, h_t^{i+1}) = \text{RNNCellMap}^i(t, L)(x_t^i, h_t^i).
$$

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)

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

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

1. if stateDim (RNNCellMap$(t - 1, L)$) != stateDim (RNNCellMap$(t, L)$), then ProjMap$(t, L) = M_t$
2. else ProjMap$(t, L) = I$

with

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

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

Therefore, the well-formed recurrent definition of the EvoRNN writes

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

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

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.

### 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.