Time Series Deinterleaving of DNS Traffic

Amir Asiaee T.
Ohio State University
Columbus, OH
asiaeetaheri.1@osu.edu

Hardik Goel
Microsoft Corporation
Redmond, Washington
hagoel@microsoft.com

Shalini Ghosh, Vinod Yegneswaran
SRI International
Menlo Park, CA
{shalini,vinod}@csl.sri.com

Arindam Banerjee
University of Minnesota
Minneapolis, MN
banerjee@cs.umn.edu

***Abstract*—Stream deinterleaving is an important problem with various applications in the cybersecurity domain. In this paper, we consider the specific problem of deinterleaving DNS data streams using machine-learning techniques, with the objective of automating the extraction of malware domain sequences. We first develop a generative model for user request generation and DNS stream interleaving. Based on these we evaluate various inference strategies for deinterleaving including augmented HMMs and LSTMs on synthetic datasets. Our results demonstrate that state-of-the-art LSTMs outperform more traditional augmented HMMs in this application domain.**

***Index Terms*—DNS, Deinterleaving, LSTM, Malicious Domain Detection**

I. Introduction

Deinterleaving temporal data streams is a general machine-learning problem with important applications to security and privacy. Specifically, interleaved network data streams are a common occurrence in cyber-threat monitoring which complicates many analyses. In many instances, the individual stream identifiers are unavailable due to technical challenges, such as the vantage point of the data collector or are intentionally supressed to protect the privacy of users in the network.

For example, consider packet traces collected in a local area network where the source IP addresses are removed, or data collected from the external-facing interface of a proxy server, or a NAT firewall where individual client identifiers are unavailable. Detecting anomalous behavior, especially stealthy and low-volume attack patterns, in these aggregated noisy streams is significantly more challenging than in a traditional deinterleaved setting.

In this paper, we discuss a variant of this problem, i.e., deinterleaving client request streams from recursive DNS resolvers to mine threat intelligence. Such DNS data streams are shared among Internet service providers (ISPs) through mediums such as the Security Information Exchange (SIE) [6] and are a valuable source of intelligence to the cybersecurity community. Here, the individual client requests to the recursive DNS resolver are typically suppressed and what we have are inter-resolver communications (i.e., communications between the recursive resolver and the root server, TLD servers and other secondary resolvers). We are interested in the application of advanced machine-learning techniques to automate the extraction of malware domain groups [6] from such resolver streams.

Malware infections while browsing the Internet have become very prevalent and occur due to various reasons such as drive-by exploits, phishing attacks etc [15], [17]. In a typical infection, the user starts from a landing page and then goes through a sequence of seemingly harmless intermediate websites, until reaching a site that contains the malicious exploit that harm the user by installing malware or stealing private data. The intermediate sites are typically redirection chains implemented in JavaScript for the purpose of obfuscation. Even though many landing and exploit websites are continously identified and blacklisted, thousands of new malicious domains emerge daily. However, pieces of the redirection infrastructure get reused across campaigns and thus the actual sequence of websites traversed by the user contains information that may help in quickly identifying new exploit sites.

When a user makes a browser request to visit a website, it first resolves the domain name by asking its recursive resolver. If the answer for the query is cached by the resolver the answer is immediately provided to the client. Otherwise, it initiates a set of recursive queries, leading to the final queried website's IP address. Each webpage may have several embedded objects from many domains leading to a sequence of domain lookup requests emanating from the client. Tracking the set of DNS requests made by each client is thus a useful means to identifying new and emergent malware infection sequences. However, to protect user privacy ISPs typically only capture data from the external facing interface of the recursive resolver, effectively suppressing the individual client stream identifiers. As there are hundreds of users making requests at the same period of time, and all of these requests are pushed to a single queue of a local DNS resolver, we cannot tell apart individual user's sequences of requests and perfectly deinterleaving all requests for deanonymization purposes is impossible. However, our objective is not deanonymization, but rather extraction of malware domain sequences which are observed repeatedly across resolvers. We believe that advanced machine learning strategies could be in such selective deinterleaving of DNS time-series for the extraction of malware domain groups.

Prior Work. To the best of our knowledge deinterleaving has not been applied to DNS resolver queue's data. Some earlier work [6] investigates the use of a sliding window approach to identify new malicious domains by exploring the domains that typically form neighbors of known malicious domains in the resolver queue, while ignoring the actual sequential information. The challenges of applying existing deinterleaving methods to DNS data is twofold. First, most of the methods has been designed for deinterleaving Markov chains [2], [12]–[14] and HMMs [10], and as we will discuss in Section II, the dynamics of submitting new queries to the local resolver is more complicated than simple Markov chain or HMM. Moreover, the state space of the models and number of sequence sources are very small in previous work applications [10], [12], while in our application, huge number of websites explodes the size of state space and also tens of users may be active in a network simultaneously. Because of the nature of our dataset, we need to use tools other than those adopted in literature [3], [4], [10], [12].

Another very useful model for time-series is Recurrent Neural Networks (RNN). Recently, RNNs and their variants (Gated Recurrent Units (GRUs) [5], Long Short-Term Memory (LSTMs) [8]) have seen a lot of success in modeling time-series in multiple domains [1], [7], [16]. However, to our knowledge even simple RNN tools have not been applied to the deinterleaving problem. Using RNN-type tools for deinterleaving mixed DNS request logs is a completely unexplored area. Motivated by the power of LSTMs to model non-linear dependencies, we seek to apply LSTMs to such data and start a new direction of work towards identifying newer malicious domains more efficiently.

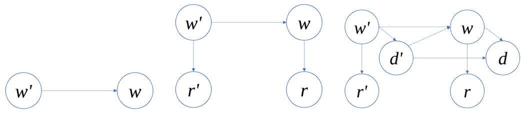
Contributions. This paper presents a preliminary exploration of the utility of various machine-learning models to address the time series deinterleaving problem for malware domain group extraction. Specifically, we present a model for DNS request generation and resolver-sequence interleaving and evaluate the utility of various inference strategies on sythetic examples including Augmented Hidden Markov Models (AHMMs) and LSTMs finding that LSTMs outperform AHMMs. Extending this analysis to real and large-scale datasets is future work.

II. Problem Formulation

A user starts by visiting a page e.g., `a.com`. While launching the webpage, many queries are being generated from different components of that browsed webpage: `a.com`, `ad1.com`, `audio1.org`. In another scenario a webpage can redirect the user to a sequence of other pages and generate sequence of requests. We refer to this sequence as a *query episode*. Next, when the user opens a new website another episode is started. The same process generates query sequences for other users. For example, a second user generates: `b.com`, `ad2.com` and after the interleaving we may observe the following sequence in the resolver:`b.com`, `a.com`, `ad1.com`, `ad2.com`, `audio1.org`. We call this process *request interleaving*. Our goal is to deinterleave the two request sequences.

A. User's Request Generation Model

The *browsing process* of a user can be modeled as simple as a Markov chain (MC) of webpages, an HMM, or an HsMM model. Figure 1 illustrates these three different user model. We model the browsing process described above using a Hidden Semi-Markov Model (HsMM). MC and HMM are special cases of this process. The hidden layer of the HsMM consists of random variables W representing the browsed webpages. Note that pages are hidden because what we see are only the DNS requests.



(a) Markov Chain (b) HMM (c) HsMM

Fig. 1: User's browsing models.

The page transition matrix is different for each user and is represented by the matrix $\mathbf{P}_u$. The observed state of the HsMM is the domain name request R which will be put in the resolver queue. Note that the time between subsequent browsed pages (which is equal to the time spend in a page before moving to the next one) in reality is different from the duration parameter in our model. In real world data, each user spends an interval on a page but in our model since we are only interested in the order of queries, we only count the number of requests that the page will query from the resolver and represent it by the random variable D. So the duration parameter D represents the number of outstanding requests from the current page.

Fig 1c shows the details of the model and Table I summarizes the model parameters. $\mathbf{O}_u(w, r)$ is the probability of submitting (outputting/observing) request r on the webpage w, for the user u. Conditional probabilities of the model are as follows:

$$\begin{aligned} \mathbb{P}_u(w|w', d') &= \begin{cases} [\mathbf{P}_u]_{w'w} & d' = 1 \\ \delta(w, w') & d' > 1 \end{cases}, \\ \mathbb{P}_u(d|w, d') &= \begin{cases} p_w(d) & d' = 1 \\ \delta(d, d'-1) & d' > 1 \end{cases}, \\ \mathbb{P}_u(r|w) &= [\mathbf{O}_u]_{w,r} \qquad , \end{aligned} \tag{1}$$

The duration parameter cannot be zero, when $d = 1$ (i.e., the page's last request is submitted) and the user moves to another page which resets the duration using $p_w(d)$. The duration probability $p_w(d)$ determines the number of requests that the webpage w will query and is independent of user u.

B. Resolver's Sequence Interleaving Model

Each time step in our model is a slot in the resolver's queue. Since there cannot be two requests in the same slot, only one user out of m can fill the t-th slot of the queue. Considering the frequency of request generation, we assume that each user i has a probability of α_i to generate the t-th request where $\sum_{i=1}^{m} \alpha_i = 1$. So if a user is very active it has higher α_i and submit requests more often.

In a more complicated setting, one can model the the "turn" of m users as a Markov chain. We name the transition matrix of the user's Markov chain as $\mathbf{A} = [\alpha_{ij}] \in \mathbb{R}^{m \times m}$. Therefore the probability of user j generating the t-th request from the i-th user is $\mathbb{P}(U(t) = j | U(t-1) = i) = \alpha_{ij}$. The random

Symbol	Explanation
W	RV for the webpage
D	RV for the number of requests to be issued on a page
R	RV for the issued DNS request
m	Number of users
n	Total number of pages
q	Maximum number of requests per page
$\mathbf{P}_u \in \mathbb{R}^{n\times n}$	Webpage transition matrix of the user u
$p_w(d), d \in [q]$	Distribution of number of requests d on the page w
$\mathbf{O}_u \in \mathbb{R}^{n\times n}$	Output distribution matrix of the user u

TABLE I: Summary of the model parameters and random variables (RV). For each random variable the corresponding small letter represents a realization. Note that W and D depend on the user but to avoid cluttering we omitted the index u.

variable $U(t) \in [m]$ represents the active user that generated the t-th request of the resolver queue. As mentioned above, a simplified variant of the *user's transition matrix* $\mathbf{A}$ is the *shares vector* $\boldsymbol{\alpha}$ that has been used in literature [2], [12] where $\forall i, j : \mathbb{P}(U(t) = i|U(t-1) = j) = \alpha_i$.

To distinguish each user's corresponding HsMM random variable in the interleaving process we use both user index and time index. For example, $W_k(t)$ is the user k's current webpage. Note that here the time is different from the real world time and HsMM duration that discussed in Section II-A. Time here is just an index into the resolver's sequence of queries. For example, $W_k(t)$ shows the webpage of user k when the tth request was submitted to the resolver.

We model the interleaving process as an Augmented Hidden Markov Model (AHMM), where the hidden states are augmented states, i.e., combination of variables [12]. To make the equations more readable, we lump together the variables corresponding to each user and make the following lumped variable $L_k(t) = (W_k(t), D_k(t))$ and the hidden state of the HMM becomes $H(t) = (L_1(t), \ldots, L_m(t), U(t))$ which is a $2m+1$ dimensional vector. Fig 2 illustrates the interleaving process that leads to sequence generation. For simplicity, we assume $u(t-1) = u'$ and $u(t) = u$ which means that users u' and u are active at time steps $t-1$ and t respectively. At the time step t, user $u(t) = u \in [m]$ generates the request $v(t)$ which is the observed (visible) variable of the HMM. The request $v(t)$ is determined by the next request of the user in its HsMM model, i.e., $r_u(t)$. Therefore, the emission probability of the AHMM is:

$$\mathbb{P}(V(t) = v(t)|H(t) = h(t)) = \mathbb{P}_u(r_u(t)|w_u(t)) = \mathbf{O}_u(w_u(t), r_u(t)).$$

Now we derive the entries of the transition probability matrix of the AHMM:

$$\mathbb{P}(H(t)|H(t-1)) = \alpha_{u'u} \prod_{k=1}^{m} \mathbb{P}(l_k(t)|l_k(t-1), u), \qquad (2)$$

In the case of $k \neq u$ the user k is not active, i.e., stalled. Substituting the probability distributions from (1), we get:

$$\mathbb{P}(l_k(t)|l_k(t-1), u) = \begin{cases} k \neq u & \delta(w, w')\delta(d, d') \\ k = u & \begin{cases} d = 0 & p_w(d)[\mathbf{P}_u]_{w'w} \\ d > 0 & \delta(d, d-1) \end{cases} \end{cases} \qquad (3)$$

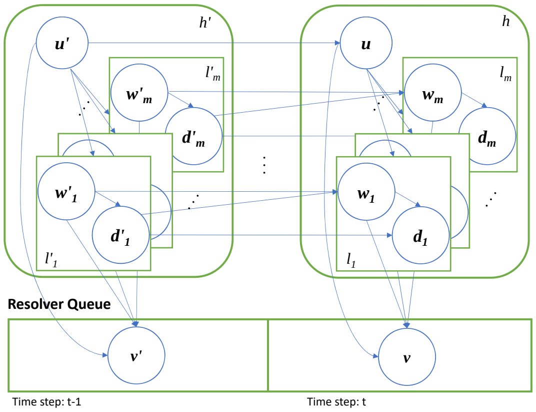


Fig. 2: Illustration of the interleaving process. The random variable u selects the user that generates the query for the time step t and stalls the others. The selected user proceed according the user model, HsMM, and outputs the query v.

Symbol	Explanation
L	The lumped random variable $L = (W, D)$.
H	The hyper-hidden state of the HMM $H = (L_1, \ldots, L_m, U)$.
V	The visible state of the HMM which is the requested DN.

TABLE II: Summary of the augmented variables. For each random variable the corresponding small letter represents a realization.

III. Deinterleaving Methods

In the deinterleaving problem, given $\{v(t)\}_{t=1}^{T}$ we are interested in inferring $\{u(t)\}_{t=1}^{T}$. In other words, we want to find the users who initiated each request from the sequence generated by the interleaving process described in Section II-B.

We present two candidate approaches for inference. One is based on reducing the interleaving process to an AHMM as discussed in Section II-B. This approach has been used for deinterleaving of Markov chains with small number of chains (users) and state space [12]. Next, we propose to deinterleave using an LSTM model which have recently been shown to perform well in many time-series analysis tasks [5], [8].

A. Inference on Augmented HMM

We can model the whole interleaving process as an AHMM and use learning techniques (like EM) to learn its parameters and use Viterbi inference to determine the most probable hidden (augmented) states $h(t)$ from which we can extract the most probable user $u(t)$. The main difficulty of applying this framework is that the state space of hidden variable, Figure 1, is very large. More specifically, there are $m(nq)^m$ possible states of h and as we increase number of webpages n or users m the state space grows exponentially. The huge state space, makes the inference and learning very hard and as we show in Section IV-A for synthetic experiments, even when the model parameters are known, deinterleaving performs (using Viterbi coding) poorly.

B. Inference using an LSTM

RNNs [11] are popular for modeling time series data. Given the input v_t and hidden state h_{t-1}, the RNN computes the next hidden state representation h_t and output u_t using the following recurrent relationships

$$h_t = f(W_v v_t + W_h h_{t-1} + b) \qquad (4)$$

$$u_t = W_u h_t \qquad (5)$$

where W_v, W_h, W_u and b are the network parameters, and $f()$ is some non-linear function. An example of f could be a sigmoid $f(z) = \sigma(z) = 1/(1 + \exp(-z))$ or rectified linear unit $f(z) = \max(0, z)$.

For our specific problem of deinterleaving, an RNN can by posed as a multi-class classification problem, where the input is the observed webpage and the output will be the identified user who requested that webpage. Specifically, each data instantiation consists of a sequence of user-request pairs, i.e., $(u(t), v(t))$. This represents who was the user at a given time t and what request was produced by that user. Both the user and the request are represented by an integer. The RNN is unrolled for the entire length of one sequence. The users and request integers are converted to one-hot encoding to enable learning. Thus if there are b possible web pages, the requests become b-dimensional vectors and for m users, it becomes an m-dimensional vector. The request vectors are fed as input to the RNN model, while the output is the corresponding user at each time-step. The RNNs m-dimensional output is passed through a softmax layer to convert it into probabilities and the user with higher probability is compared against the ground truth. Performance is measured in terms of accurately identifying the user at each instant.

A common variant of RNN is LSTM [8] which we use in our experiments. We randomly initialize network parameters W_v, W_h, W_u and apply stochastic gradient descent (SGD) (for RNNs, it is also referred to as a Backpropagation-through-time (BPTT) algorithm). In particular, we use a variation of the standard SGD called Adam [9], which allows for adaptive learning rates using the past gradients, similar to using momentum. This results in faster convergence compared to other adaptive algorithms.

IV. Experiment

We start with a synthetic toy example and compare our LSTM algorithm with Viterbi inference as the baseline and then move to larger experiments. In all experiments, accuracy is measured as $\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}(u_t = \hat{u}_t)$ where u_t is the actual user generated query t and $\hat{u}_t$ is the inferred user.

A. Viterbi vs. RNN

Here we generate synthetic resolver queue using the most complicated user model, i.e., HsMM of Figure 1c and report the Viterbi and LSTM methods performance. To reduce the computational burden for the Viterbi algorithm, we restrict ourselves to 2 pages, 2 users, and 2 possible requests per page. To make the setup even simpler, user i browse only page i and page i picks from two possible requests at random using Beta(3+ϵ,1+δ) where ϵ and δ are independent and uniform over $[0, 1]$. Viterbi is tested on the same sequences of size thousand. Results are averaged over 5 realizations of the synthetic data. With this setup the size of the hidden state space of the AHMM built from the HsMM user model is 32 and the number of observations is 2. The users shares vector is $\boldsymbol{\alpha} = (.4, .6)$. LSTM is trained, validated and tested with sequences of size 6, 3, 1 thousands requests, respectively.

Method	Viterbi	LSTM
Mean Accuracy	0.51	0.92
Std of Accuracy	0.02	.17

TABLE III: Comparing accuracy of Viterbi coding and LSTM methods for the toy example. Results are averaged over 5 realization of the synthetic data. The baseline accuracy based on the proportion of users $\boldsymbol{\alpha} = (.4, .6)$ is .6.

Parameter	Value
m	2 users
n	20 pages
q	Maximum of 5 request per page
$\boldsymbol{\alpha}$	$(0.4, 0.6)$
$\mathbf{A}$	Diagonal dominated row stochastic random matrix*.
$\mathbf{P}_u$	A random 20×20 matrix*
$p_w(d)$	Uniform$(1, 5)$
$\mathbf{O}_u$	A random 20×20 matrix*

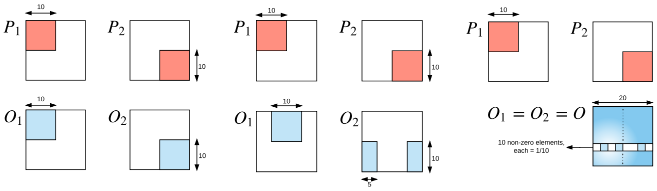
TABLE IV: Summary of the experimental setup for the synthetic experiment IV-B. *More on the random matrix generation in the text.

Table III summarizes the result: Interestingly, LSTM outperforms Viterbi by a large margin. Note that we perform Viterbi assuming that HMM parameters are given and not learned from data using algorithms like Baum–Welch, and even with this setup Viterbi performs poorly, worst that the baseline. Perhaps, the poor performance of Viterbi compared with LSTM can be explained by the linear nature of Viterbi coding and the intrinsic power of LSTM in learning non-linear temporal relations.

B. Synthetic Experiment

Owing to the poor performance of AHMM approach from now on we focus on LSTM method of Section III-B. We report the results of seven synthetic experiments only for LSTM which is trained, validated and tested with sequences of size 60, 30, 10 thousands requests, respectively.

We test the results for 7 different scenarios, in all of them we want to deinterleave a sequence generated by two users but the parameters in each experiment is set up differently. Table IV specifies the shared parameter setup. Specific user transition and emission matrices are set for different scenarios which are explained in Section IV-B. Note that in our experiments we report results on two set of synthetic data set, where in one we have a users shares vector $\boldsymbol{\alpha}$ determining the share of each user from the queue's requests. In the other more general data generating scheme, we assume that the users transition



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 3: Illustration of the disjoint surfing categories for $a = 10$ and $b = 20$.

matrix $\mathbf{A}$ governs the turn in request submission. Different distributions for $\boldsymbol{\alpha}$ and $\mathbf{A}$ are discussed in Section IV-B.

Sparsity Patterns of Matrices: For each user u we have two matrices $\mathbf{P}_u$ and $\mathbf{O}_u$ which are randomly generated. The generation process assumes that each row of both matrices is sparse, which is a reasonable assumption. Each user view and surf a limited number of pages and on each page the possible requests are from a small subset of the all available pages. The supports of $\mathbf{P}_i$s and $\mathbf{O}_i$s can overlap or be disjoint and this combination generates the different setups of our experiments. After selecting a support we generate a discrete distribution over that support, which will be discussed in Section IV-B.

In the following the outer-list determines the different strategies for generating $\mathbf{P}_u$s and the inner-list elaborates the method of building $\mathbf{O}_u$s. Each row of $\mathbf{O}_u$s has a non-zero elements (randomly selected) and the distribution is uniform. We call $\mathbf{O}_1 \neq \mathbf{O}_2$ and $\mathbf{O}_1 = \mathbf{O}_2$ schemes, personalized and shared outputs respectively.

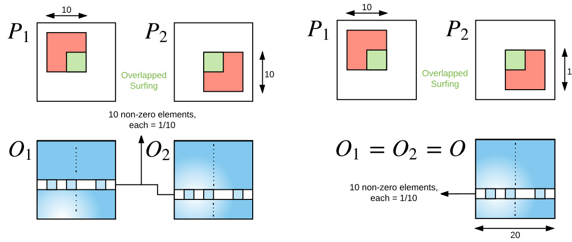
- **Disjoint webpage surfing:** In this scenario, users surf disjoint parts of the web, say user 1 surf inside a group of first a pages and user 2 surf the remaining $n-a$ pages, Fig 3.

  Case 1) *Disjoint personalized outputs - same grouping as webpages:* $\mathbf{O}_u$ and $\mathbf{P}_u$ have similar sparsity patterns, Fig 3a.
  Case 2) *Disjoint personalized outputs:* $\mathbf{O}_u$ and $\mathbf{P}_u$ do not have similar sparsity patterns, but support of $\mathbf{O}_1$ and $\mathbf{O}_2$ are disjoint, Fig 3b.
  Case 3) *Shared output:* Fig 3c.
- **Overlapped webpage surfing with fixed block size:** Each user selects its surfing support of size a at random. Supports may overlap, Fig 4.

  Case 4) *Personalized outputs:* Fig 4a.
  Case 5) *Shared output:* Fig 4b.
- **Overlapped webpage surfing with variable block size and interaction between blocks:** Each user selects $s = \text{Uniform}(1, a)$ pages at random as its main support (higher probability of surfing in these s pages), and $a-s$ pages again at random as its auxiliary support (pages that user seldom visits), Fig 5.

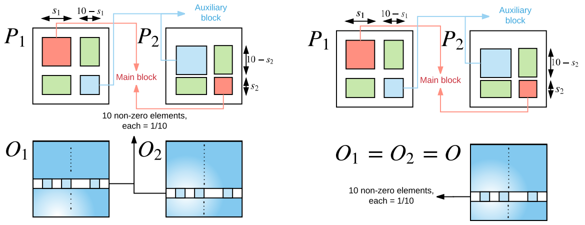
  Case 6) *Personalized outputs:* Fig 5a.
  Case 7) *Shared output:* Fig 5b.

Probability Distributions: Here we explain different distributions used in our synthetic generator:



(a) Case 4 (b) Case 5

Fig. 4: Illustration of the overlapped surfing without auxiliary block for $a = 10$ and $b = 20$.



(a) Case 6 (b) Case 7

Fig. 5: Illustration of the overlapped surfing with auxiliary block for $a = 10$ and $b = 20$.

- Users shares vector $\boldsymbol{\alpha}$: We fix $\boldsymbol{\alpha}$ to $(.4, .6)$.
- Rows of user transition matrix $\mathbf{A}(u)$: This is a diagonal dominant matrix, meaning that if user u has submitted the current request $v(t)$ it is more probable that he submit the next request. In this way, we capture the fact that because of the episodic nature of the request submission, close-by queries are more probable to come from a same user. This has been exploited in the previous literature [6]. Note that when instead of matrix $\mathbf{A}$ we only consider the vector $\boldsymbol{\alpha}$ we may not capture this realistic property of the data. For the toy example, we set $\alpha_{ii} = .5 + \frac{1}{2}\text{Uniform}(0, 1)$ and $\forall i \neq j : \alpha_{ij} \propto (1 - \alpha_{ii})\text{Uniform}(0, 1)$
- Rows of output matrix $\mathbf{O}_u(w)$: As mentioned before, each row $\mathbf{O}_u(w)$ has a non-zero elements with each with probability $1/a$.
- Rows of page transition matrix $\mathbf{P}_u(w)$: In a nutshell, we discretize a continuous Beta distribution with different parameters for each block and normalize the final vector. The distribution that we use for the (main) support is Beta(3+ϵ,1+δ) where ϵ and δ are random numbers from $[-1, 1]$. For the distribution on the auxiliary support of the cases 6 and 7 above, we use Beta(2+ϵ,2+δ).

Discussion: Table V shows the error of our method for all 7 cases of Section IV-B for $a = 10$.

Each row is the average result for five instantiation of the model parameters $\mathbf{O}_u$ and $\mathbf{P}_u$. The error of each instantiation (each row) is an average of 100 experiments. Note that case 1 and 2 are trivial cases when both $\mathbf{P}$s and $\mathbf{O}$s are disjoint and LSTM perfectly dis-interleave. Interestingly, performance in case 3 is much worse than cases 1 and 2, which confirms

Cases		1	2	3	4	5	6	7
$\boldsymbol{\alpha}$	Mean	1	1	.63	.70	.62	.74	.65
	Std	0	0	.02	.02	.02	.05	.04
A	Mean	1	1	.77	.69	.67	.82	.78
	Std	0	0	.10	.09	.08	.09	.16

TABLE V: Deinterleaving accuracy of LSTM for different cases of the synthetic example when user transitions are determine by either of $\boldsymbol{\alpha} = (.4, .6)$ or random diagonally dominant $\mathbf{A}$. The baseline for $\boldsymbol{\alpha}$ and $\mathbf{A}$ experiments are .6 and .5 respectively.

that in our model having disjoint output matrices is more important than disjoint surfing pattern. Intuitively, this makes sense because the final request comes from the output matrices and if we have personalized outputs the deinterleaving should be easier. Interestingly, beyond the trivial cases 1 and 2, case 6 has the best accuracy, probably because of personalized outputs and more complicated $\mathbf{P}_u$ for each user (composed of main and auxiliary block) makes the whole problem more separable.

V. Conclusion

This paper describes our foray into the application of advanced deep-learning techniques to the problem of deinterleaving DNS-based time-series sequences. To this end, we developed an HsMM-based model of user request generation and an AHMM-based model of the interleaving process at the resolver queue. We then evaluated the efficacy of two different inference strategies for deinterleaving on a synthetic dataset. Our results suggest that LSTM-based strategies significantly outperform traditional AHMM-based models. In future work, we plan to extend this analysis on signficantly larger datasets to the specific problem of malware domain group extraction.

Acknowledgments

The work was supported in part by NSF grants CNS-1314560, IIS-1447566, IIS-1447574, IIS-1422557, CCF-1451986, and IIS-1563950. SG and VY acknowledge partial support from NSF Grant CNS-1314956 and CNS-1514503.

References

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

[2] Tugkan Batu, Sudipto Guha, and Sampath Kannan. Inferring mixtures of markov chains. In *COLT*, volume 2004, pages 186–199. Springer, 2004.

[3] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic dna. *Journal of molecular biology*, 268(1):78–94, 1997.

[4] Christopher B Burge and Samuel Karlin. Finding the genes in genomic dna. *Current opinion in structural biology*, 8(3):346–354, 1998.

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

[6] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. An empirical reexamination of global dns behavior. *ACM SIGCOMM Computer Communication Review*, 43(4):267–278, 2013.

[7] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552. 2009.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Niels Landwehr. Modeling interleaved hidden processes. In *Proceedings of the 25th international conference on Machine learning*, pages 520–527. ACM, 2008.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[12] Ariana Minot and Yue M Lu. Separation of interleaved markov chains. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, pages 1757–1761. IEEE, 2014.

[13] Gadiel Seroussi, Wojciech Szpankowski, and Marcelo J Weinberger. Deinterleaving markov processes via penalized ml. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1739–1743. IEEE, 2009.

[14] Gadiel Seroussi, Wojciech Szpankowski, and Marcelo J Weinberger. Deinterleaving finite memory processes via penalized maximum likelihood. *IEEE Transactions on Information Theory*, 58(12):7094–7109, 2012.

[15] SophosLabs. Looking ahead: Sophoslabs 2017 malware forecast. 2017.

[16] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[17] Symantec. 2017 internet security threat report. 2017.