





Generation of Realistic Navigation Paths for Web Site Testing Using Recurrent Neural Networks and Generative Adversarial Neural Networks

Silvio Pavanetto^(✉)  and Marco Brambilla^(✉) 

Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano, P.za L. da Vinci 32, Milano, Italy
{silvio.pavanetto,marco.brambilla}@polimi.it

Abstract. A robust technique for generating web navigation logs could be fundamental for applications not yet released, since developers could evaluate their applications as if they were used by real clients. This could allow to test and improve the applications faster and with lower costs, especially with respect to the usability and interaction aspects. In this paper we propose the application of deep learning techniques, like recurrent neural networks (RNN) and generative adversarial neural networks (GAN), aimed at generating high-quality weblogs, which can be used for automated testing and improvement of Web sites even before their release.

Keywords: Web engineering · Data mining · Deep learning · Recurrent neural networks · Generative adversarial networks · Testing

1 Introduction

Weblogs represent the navigation activity generated by a specific amount of users on a given website. This type of data is fundamental, e.g. for a company, because it contains information on the behaviour of users and how they interface with the company's product itself (website or application). The first useful information that can be extracted from weblog is the quality of the website, as described in the work of Berendt and Spiliopoulou [2], where they try to understand navigation patterns that are present in the data. This is explained also in the work of Singh et al. [15] in 2013, that shows an overview of the web usage mining techniques by applying pattern recognition. In addition, one could analyze these patterns and the statistics about users activities with visualization tools as explained in the work of Bernaschina et al. [3].

If a company could have a realistic weblog before the release of its product, it would have a significant advantage because it can use the techniques explained above to see the less navigated web pages or those to put in the foreground, but users and time are needed to produce them, making it an expensive task.

Because of this limit, our focus is on the generation part, since this particular task is little explored, but it is often a recurring theme also in the research world due to the lack of publicly available data.

In fact, open source libraries like Flog Generator [9] and Fake Apache Generator [1], or the work by Lin et al. [10], generate logs in a random manner and cannot be used as datasets that represent the users behaviour. Therefore, being able to create an algorithm that generates high-quality weblogs would be relevant from a scientific and commercial point of view.

What we did was apply deep learning methods for generating more realistic navigation activities, starting from a RNN (Hochreiter Sepp and Jürgen Schmidhuber [8]), which has been seen that it can be used for generating complex sequences with long-range structure (Alex Graves et al. [7]). Then trying a GAN (Goodfellow et al. 2014 [5]): neural networks aimed at generating new data, such as images or text, very similar to the original ones and sometimes indistinguishable from them, that have become increasingly popular in recent years.

The challenge is to evaluate which algorithm for the generation of log data could be the best and to verify if the GAN is applicable to this problem. Our work starts with the implementation of a generative algorithm based only on the theories already presented in the literature concerning the analysis and generation of weblogs. Then we introduce the algorithms that falls into the category of deep learning: an RNN and a GAN, verifying the effective generative capacity of these neural networks.

This paper is structured as follows: first, we talk about the state of the art for web mining and discrete data sequences generation topics. Then, there is a section about the methods used in this work, followed by an implementation and experiments part. Lastly, we close with the conclusions and future works.

2 Related Work

Berendt and Spiliopoulou [2] in 2000 have demonstrated the appropriateness of the “Web Usage Miner” (WUM): a set of tools which discovers navigation patterns subject to advanced statistical and structural constraints. This work was intended to understand the quality, defined as the conformance of the web site’s structure to the intuition of each group of visitors accessing the site, of a specific website as a whole and not considering every page as a single. For doing this, they used data mining techniques such as sequence pattern mining and apriori algorithm.

The work of Singh et al. [15] in 2013 shows an overview of the web usage mining technique by applying pattern recognition on weblog data, defined as the act of taking in raw data and making an action based on the ‘category’ of the pattern. They divide their work into three parts: Preprocessing, Pattern discovery and Pattern analysis.

Generally speaking, these works that analyse web log data for pattern discovery, use almost the same approach based on data pre-processing and data

mining techniques previously explained, in addition to a clusterization in some cases (Vedaprakash et al. [16] and Mahoto et al. [11] are examples).

Regarding weblog data generation, the open source malicious log detection library [10] tries to generate new access log data, by inserting in some malicious activities with the purpose of identifying them. The problem with [10] and other open source libraries such as Flog Generator [9] or Fake Apache Generator [1], is that they create these logs in a random manner. Instead, in this work we produce them in a completely different and more structured way. With respect to deep learning techniques used to produce discrete data, we can start with LSTM in recurrent neural networks (RNNs), presented for the first time by Hochreiter Sepp and Jürgen Schmidhuber [8]. This type of RNNs was widely used in the subsequent works, like the work by Alex Graves [7] that shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. Their approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). Due to the success of this type of neural network applied to data sequences (real-valued and discrete ones), this work proposes a Long Short-term Memory recurrent neural networks approach as the first deep learning method.

In the past few years, new techniques have been presented for generating high-quality data; the most famous and promising is the GAN (Goodfellow et al. 2014 [5]) that uses a discriminative model to guide the training of the generative one. However, it has limitations when the goal is for generating sequences of discrete tokens. A major reason lies in that the discrete outputs from the generator make it difficult to pass the gradient update from the discriminative model to the generative model. In addition, the discriminative model can only assess a complete sequence, while for a partially generated sequence, it is non-trivial to balance its current score and the future one, once the entire sequence has been generated. Yu et al. [17] try to solve this problem, proposing a sequence generation framework, called SeqGAN. Modeling the data generator as a stochastic policy in reinforcement learning (RL), SeqGAN bypasses the generator differentiation problem by directly performing gradient policy update. The RL reward signal comes from the GAN discriminator judged on a complete sequence and is passed back to the intermediate state-action steps using a Monte Carlo search. However, in their work they use a ‘oracle’ model, that is a randomly initialized LSTM as the right model, to generate the real data distribution $p(x_t|x_1, \dots, x_{t-1})$ for their experiments and evaluations. In this way, they have a significant benefit: it provides the training dataset and then evaluates the exact performance of the generative models. In our approach we use real data as training data instead, and at the end of the GAN training we evaluated the results with different metrics.

3 Deep Learning Based Log Generation

In this section we present our statistical and deep learning approaches for generating weblogs. The core idea is to develop a recurrent neural network and a

generative adversarial network (GAN) for generating new weblog data, and compare the generation performance of these methods with the statistical algorithm.

3.1 Statistical Approach

To this day the only public libraries simply create logs in a completely random manner. This approach is very coarse and therefore we decided not to consider it even as a baseline. We propose instead a method composed by two main parts: the first analyses a website and extracts statistical information, the second uses those data for generating new weblogs. The input must contain some important elements, such as the *Entry Points*¹, the *Confidences*², the *Mean Times*³ and the *Web Site Graph*⁴.

The implementation uses the state of the art methods for the extraction of knowledge from logs and applies this information on the creation process. Regarding the actual implementation of this algorithm, first we need to set different variables to produce new logs, like the *Maximum number of IPs at the same time*, a *List of users IPs*, the *Number of navigation sessions* and so on. Once all the configuration parameters are set, the algorithm can start the generation part where, if S is the total number of navigation sessions previously established, it repeats S times the computation of the *Navigation Path*, that consists of:

- *Selection of Entry Point*: The first thing that the algorithm needs to compute for every navigation session, is the entry point of the sequence. This page is selected among all the home pages that has been received as input by the algorithm, using the associated probability of being selected. It is not possible to start the navigation with a page that is not present on the home pages list.
- *Computation of Next URL*: After the selection of the entry point, the algorithm chooses the next URL until the sequence length is reached: this is the exit condition of every loop iteration. This URL is selected by retrieving all the possible subsequent pages for the previous computed URL and then by picking one of them using the probability of moving from one page to another.
- *Computation of Residence Time*: After each couple of URLs is chosen, it is necessary to calculate the number of seconds that the user will spend on page A before moving to page B or terminating the navigation. This is done by looking at the *Mean Times* that come as input to the algorithm and picking the mean time that corresponds to that couple of pages.

¹ A list of pages that represent the possible entry points for every navigation session. A probability to start the navigation with that page is associated with each one.

² The confidences are the probabilities for moving from a specific page to another, or, the probabilities for moving to a new page at a particular moment T , knowing the complete navigation path done from the beginning of the session (In this case, session means a portion of continuous time in which the user is browsing without leaving or interrupt the navigation.), until T .

³ A list of mean times expressed in seconds that correspond to the quantity of time that users spend on that page on average.

⁴ The graph representing the entire web site, where each page is associated with a list of possible subsequent pages.

Once the loop cycle is completed and all the sequences have been generated, the algorithm produces a log file that contains all the navigation activity of the users, created previously. The requests are sorted by time, and then the file is created. As we illustrated, this first statistical algorithm is guided with constraints such as the probabilities of moving from one page to another during the navigation of every user and the residence time on each page, that are already computed when the algorithm starts its execution. Instead, Deep Learning techniques do not need any hand-designed feature extraction phase, because they empower the model with the capability of learning features optimized for the task at hand.

3.2 RNN-Based Approach

Among the deep learning algorithms, we chose the Recurrent Neural Networks [14] (Rumelhart et al. 1986), that are neural networks dedicated to the processing of sequential data. Simple RNNs are useful when the temporal dependencies to be learned are not too long. When this happens, the gradients propagated over many stages tend to vanish (most of the time) or explode (more rarely). Even if we consider stable structures with a reasonable number of parameters, long-term dependencies lead to exponentially smaller weight updates for long-range interactions compared to the short-term ones. The best solution to this problem found as of today are gated RNNs, which are based on creating paths through time that have derivatives that neither vanish nor explode. One of the most effective models employing gated units is Long Short-Term Memory (LSTM) [8].

Due to these features regarding Recurrent Neural Network and their memory capacity, we implemented an RNN that receives a list of navigation sessions as input and trains itself with them. After the training phase, the network is ready to predict and produce new sequences.

Unlike the statistical algorithm, we do not need to specify the probability of moving among pages. For this reason, the input for the recurrent neural network consists of a list of sequences of URLs, together with the seconds of permanence on that page (*secInPage*) and the index that represents the number of pages already visited in the same session (*indexSession*). Every sequence corresponds to a navigation session made by a specific user.

The main characteristic that we want our RNN to learn is the sequence of pages that a specific user will visit and in which order he will make his navigation. For this reason, we started by feed the network with only the *url* feature, then we added the *secInPage* and *indexSession* features. We come up with the architecture shown in Fig. 1, where we can see that there are two principal layers, composed by the *CuDNNLSTM* previously discussed. Each of these layers consists of 50 neurons and is followed by a dropout operation that avoids overfitting. The output of the second layer, after the dropout, is flattened to obtain a single 2D vector containing the inputs for the last layer: the Dense layer, which produces the final output of the network.

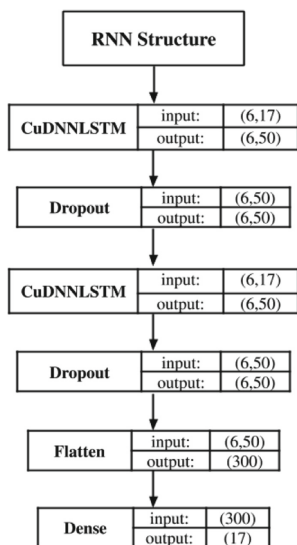


Fig. 1. The structure of the RNN. Parameters are set as follows: length of every sequence is 6, number of classes is 17, number of neurons is 50 and number of data point after the flatten operation is 300.

3.3 GAN-Based Approach

As discussed in the previous section, in the task of generating sequential synthetic data that mimics the real one, recurrent neural networks with long short-term memory (LSTM) cells have shown excellent performance. The most common approach to training an RNN is to maximize the log predictive likelihood of each valid token in the sequence given the previously observed tokens. However, the maximum likelihood approaches suffer from exposure bias in the inference stage: the model generates a sequence iteratively and predicts next token conditioned on its previously predicted ones that may never be observed in the training data. Such a discrepancy between training and inference can incur accumulatively along with the sequence and will become prominent as the length of sequence increases.

Generative Adversarial Network (GAN) proposed by Goodfellow and others [6] is a promising framework for alleviating the above problem. Specifically, in GAN a discriminative net D learns to distinguish whether a given data instance is real or not, and a generative net G learns to confuse D by generating high-quality data. This approach has been successful but has been applied almost only in computer vision tasks of generating samples of natural images (Denton et al. 2015 [4] is an example).

For these reasons and because of his capability of learning the probability distribution of training data and his hidden features, we thought that trying to build a GAN that generates synthetic discrete data would be an interesting challenge

and a useful work for understanding if these type of Neural Networks are adaptable also to this task. Unfortunately, applying GAN to generating sequences has two problems. Firstly, GAN is designed for generating real-valued, continuous data but has difficulties in directly generating sequences of discrete tokens, such as texts or URLs in our case.

As such, the gradient of the loss from D w.r.t. the outputs by G is used to guide the generative model G (parameters) to slightly change the generated value to make it more realistic. If the generated data is based on discrete tokens, the “slight change” guidance from the discriminative net makes little sense because there is probably no corresponding token for such slight change in the limited dictionary space.

Secondly, GAN can only give the score/loss for an entire sequence when it has been generated; for a partially generated sequence, it is non-trivial to balance how well as it is now and the future score as the entire sequence.

GAN Parametrization. For the development of this net, the input data is a list of sequences of URLs, encoded as integers. Every sequence in this dataset corresponds to a navigation session like the RNN input case, and has a pre-fixed length.

Looking deeper at the implementation of the GAN, the sequence generation problem is denoted as follows: Given a dataset of real-world structured sequences, train a θ -parameterized generative model G_θ to produce a sequence $Y_{1:T} = (y_1, \dots, y_t, \dots, y_T)$, $y_t \in Y$, where Y is the vocabulary of candidate URLs. This is interpreted as a reinforcement learning problem. In time-step t , the state s is the current produced URLs (y_1, \dots, y_{t-1}) and the action a is the next URL y_t to select. Thus the policy model $G_\theta(y_t|Y_{1:t-1})$ is stochastic, whereas the state transition is deterministic after an action has been chosen, i.e. $\delta_{s,s'}^a = 1$ for the next state $s' = Y_{1:t}$ if the current state $s = Y_{1:t-1}$ and the action $a = y_t$; for other next states s'' , $\delta_{s,s''}^a = 0$.

Additionally, we also train a ϕ -parameterized discriminative model D_ϕ to provide a guidance for improving generator G_δ . $D_\phi(Y_{1:T})$ is a probability indicating how likely a sequence $Y_{1:T}$ is from real sequence data or not. The discriminative model D_ϕ is trained by providing positive examples from the real sequence data and negative examples from the synthetic sequences produced by the generative model G_θ . At the same time, the generative model G_θ is updated by employing a policy gradient and MC search based on the expected end reward received from the discriminative model D_ϕ . The reward is estimated by the likelihood that it would fool the discriminative model D_ϕ .

Also, while the generator improves, we need to re-train periodically the discriminator to keep a good pace with the generator. Also, to reduce the variability of the estimation, we use different sets of negative samples combined with positive ones.

GAN Structure. Lastly, we want to add some information about the two neural networks structure that compose the GAN:

- *Generator*: We used a recurrent neural network as the generative model.
- *Discriminator*: In this case, we choose the CNN as our discriminator because these types of networks has been shown off great effectiveness in text classification, and our task is very similar to that one. A kernel applies a convolutional operation to a window of words to produce a feature map. At the end of this phase, a max-over-time pooling operation is applied over the feature maps. To enhance the performance, we used a fully connected layer with sigmoid activation that outputs the probability that the input sequence is real.

4 Evaluation

4.1 Context and Dataset

The evaluation methods and the algorithms employed in this work use the public 1995 NASA Apache web logs [12]. This public dataset is a standard Apache web log file. A typical configuration for the access log, that also applies in this case, is the Apache standard syntax for the HTTP requests. This standard format can be produced by many different web servers and read by many log analysis programs. The log file entries produced will look like the following (that is the standard apache format⁵): *127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326*

This dataset was selected for its size, number of entries and because is one of the few publicly available web log files: the lack of open web log data is in fact one of the issues that this work tries to solve. In particular, the size is 205.2 MB that corresponds to data recorded from Jul 01 to Jul 31(1995) and the total number of rows is 1891697.

URL Depth Problem. One of the main critical aspects to manage concerns the depth of every URL to be kept, present in each request. That is, if we have, for example, a request with a URL like this:

/home/shuttles/1969/apollo_11.html

we can notice that there are 4 steps in the request link: *home - shuttles - 1969* and *apollo_11*. Every step in this navigation path that leads to the *apollo_11.html* page represents a folder or eventually a category of the website that goes from the home page, that is the page with less specificity, to the page of the shuttle “apollo11”, that is specific for that type of shuttle. This is a common conceptual representation of pages in every website, but for the task of this work, represents a problem whose complexity increases exponentially in certain situations.

For clearing this concept, is helpful looking at the Table 1, where we can see how fast the number of different pages in the website grows with the URL depth variable. In fact, we have only 19 unique pages when only a single part of the URL is kept, while we have almost ten times this number with only one more depth level.

⁵ <http://httpd.apache.org/docs/1.3/logs.html>.

Table 1. The numbers of different pages with respect to the URL depth variable

URL depth	Number of pages
1	19
2	115
3	275
4	402

The problem with the algorithms that we tried to develop is that every URL is seen as a category and the neural networks are asked to predict the next page in a sequence among all. This means that if in the first case (depth = 1) the networks have to learn 19 different categories, in the last one (depth = 4) the network will have to learn 402 categories, and this is feasible only with a huge amount of training data, that is not available for our work.

Metrics. For evaluating the quality and realism of log produced by the different methods, we used a metric called the BLEU score [13]. BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations, or also, is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is the correspondence between a machine’s output and that of a human.

Every URL is treated as a unique “word” in the vocabulary, composed of all the pages of a particular website. Using this metric, scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation’s overall quality. Transferring this to our case, the translated segments are the generated navigation sequences, while the good quality reference translations correspond to our original dataset: the NASA weblog.

4.2 Experiments with RNN

The framework used for the implementation of the network is Keras, a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano, while the type of LSTM cell is CuDNNLSTM: This type of LSTM cell must be run on GPU and is based on cuDNN, developed by Nvidia. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

For evaluating the performance of the network, we trained it on a training set, and then we checked the prediction accuracy on a test set. The problem with this type of evaluation, in this case, is that in the test data we could encounter some URLs that were not seen in the training phase, and the results would not

Table 2. RNN experiments: hyper-parameters tuning, URL Depth = 1

#test	1	2	3	4	5
Length sequence	6	6	6	6	6
Neurons	50	50	20	50	40
Layers	2	3	2	4	3
Dropout	0.2	0.25	0.25	0.25	0.2
Shuffle	True	True	True	True	True
Batch size	30	30	30	20	40
Activation	Softmax	Softmax	Softmax	Softmax	Softmax
Optimizer	Adam	Adam	Adam	Adam	Adam
Loss	cat. cross-ent.	cat. cross-ent.	cat. cross-ent.	cat. cross-ent.	cat. cross-ent.
Metrics	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
Epochs	50	50	65	70	100 (early stop)
Average accuracy	74,13%	74,17%	74,69%	74,76%	74,76%

Table 3. RNN experiments: BLEU performance and best accuracy with respect to the URL Depth

URL Depth	#classes	BLEU	Best accuracy
1	19	0.6482	74,76%
2	115	0.4739	58,23%
3	275	0.3655	31,05%

have been accurate because the network could not learn something that it has never seen. For this reason, we split the data in training and test by checking that all the URLs in the test set would be present also in the training set. In addition to this, we adopted some techniques to avoid overfitting, such as *dropout*, *early stop training*, and *data shuffle*.

In the Table 2 the results of the hyper-parameters tuning are visible for the URL depth equal to one, while in Table 3 we can see the evaluation results in terms of BLEU and best accuracy, with respect to the URL depth.

As mentioned before, the URL Depth problem is crucial because it increases the complexity of learning the correct features and the network performs worse.

4.3 Experiments with GAN

In this algorithm, the training set for the discriminator is comprised by the generated examples with the label 0 and the instances from the training set with the label 1. Dropout and L2 regularization are used to avoid over-fitting. Also in this case, we tried to generate new sequences using three different URL depth level to understand how the GAN performs respect to this parameter.

In this algorithm, the most important parameters to tune are the number of training epochs for the generator and the discriminator. In fact, we noticed that if the RNN (generator) is not sufficiently pre-trained before starting the adversarial training, the generator improves quite slowly and unstably. The reason is that

in this GAN, the discriminative model provides reward guidance when training the generator and if the generator acts almost randomly, the discriminator will identify the generated sequence to be unreal with high confidence and almost every action the generator takes receives a low (unified) reward, which does not guide the generator towards a good improvement direction, resulting in an ineffective training procedure.

This indicates that in order to apply adversarial training strategies to sequence generative models, a sufficient pre-training is necessary. For the evaluation of this algorithm, we started with the analysis of the generator loss, relating it to the URL depth and to the number of pre-training epochs of the generator before the adversarial training. We run the training with three different values of pre-train generator epochs and three different values of URL Depth. The results that emerge from these analyses are the following:

- Adding a discriminator to the RNN allows the GAN to lower the loss of the generator and to improve its limits.
- Increasing the value of URL Depth variable, the loss value also increases regardless of the generator pre-train epochs value. This is observable in the Figs. 2 and is a further confirmation that increasing the number of URLs is critical for the complexity of the computations that the network must do.
- The variance of the loss in all the cases decreases when the number of pre-train epochs increases. In the Table 4, is notable that the minimum variance value occurs when the generator is pre-trained with 100 epochs and this is valid for the 3 different URL Depth values. This correspond to a better stability of the generator with respect to the cases with lower pre-train epochs.
- The minimum value of the loss is reached with 15 pre-train epochs, when the depth is equal to 1 and 2, while with a depth equal to 3, the minimum value is obtained with 100 pre-train epochs. This means the generator can get the lowest loss value with few epochs, but a good stability of the network is reached only with a high number of pre-train epochs.

We generated 3 different sets of sequences with respect to the URL depth and to the number of pre-train epochs for the generator and we computed the BLEU score against the original set of sequences. The results are shown in Table 6. We can see that adding a discriminator to the RNN (the generator) improves the scores in each of the 3 cases only if the number of pre-train epochs for the generator is enough to make the generator robust. If we train the generator only for 40 epochs and then we start the adversarial training, the data generated by the RNN will receive always a low score as a reward by the discriminator.

This is in contrast with the assessments previously made, where we showed that the lowest loss values are reached with 15 or 40 pre-train epochs, but agrees on what concerns the stability of the network that is improved with 100 pre-train epochs. This demonstrates that in the case of generative models the analyses of pure loss are not enough to understand if these models produce high-quality data (Table 5).

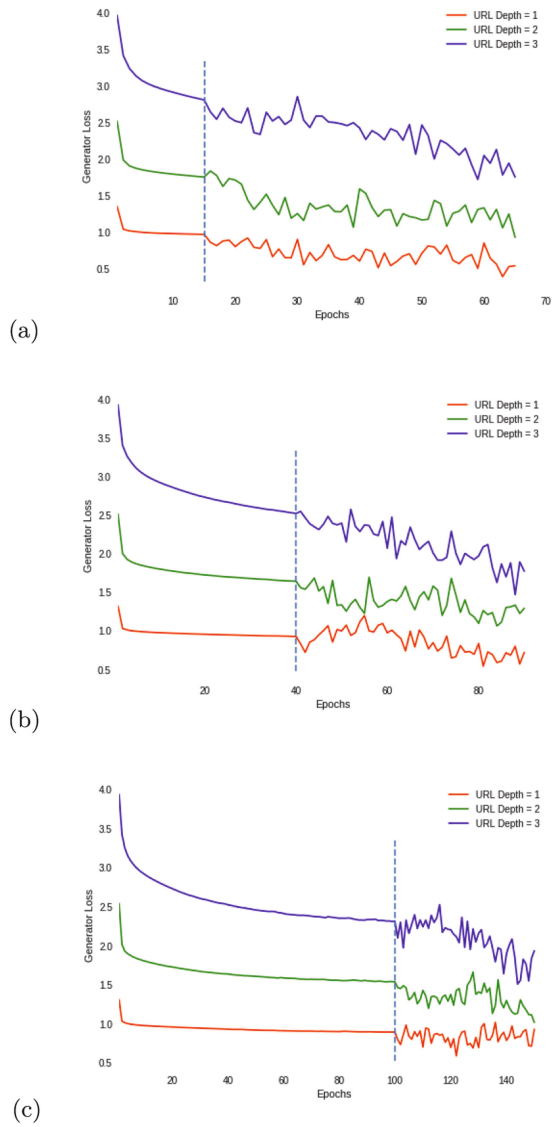


Fig. 2. Generator loss with 15, 14, and 100 pre-training epochs respectively, in relation to different URL Depth.

Table 4. GAN experiments: variance of the generator loss, related to the URL Depth and the pre-train epochs

	URL Depth = 1	URL Depth = 2	URL Depth = 3
15 Epochs	0.0151	0.0324	0.0673
40 Epochs	0.0255	0.0239	0.0614
100 Epochs	0.00832	0.0179	0.0550

Table 5. GAN experiments: minimum value of the generator loss, related to the URL Depth and the pre-train epochs

	URL Depth = 1	URL Depth = 2	URL Depth = 3
15 Epochs	0.3916	0.9308	1.7240
40 Epochs	0.5451	1.0667	1.4720
100 Epochs	0.5922	1.0215	1.5121

Table 6. GAN experiments: BLEU performance with respect to the URL Depth

URL Depth	BLEU, 40 pre-train epochs	BLEU, 100 pre-train epochs
1	0.6071	0.7243
2	0.4328	0.5471
3	0.3321	0.4839

5 Comparison: Statistical Approach vs RNN vs GAN

For the final comparison between all the techniques explored in this work, we opted to use another metric in addition to **BLEU**, that is a human judgment, since a weblog is a composition of navigation sequence and every sequence is something that is decided and created by a human. For this reason, we chose 5 of our colleagues with the same skills and knowledge: we showed him all the pages of the website and the possible navigation paths. Specifically, we mix 50 real sequences and 50 generated from GAN and RNN.

Then the judges are invited to pronounce whether each of the 100 sequences is created by human or machines. Once regarded to be real, it gets +1 score, otherwise 0. Finally, the average score for each algorithm is calculated. The experiment results are shown in Table 7, from which we can see the significant advantage of GAN over the RNN and Statistical method in weblog generation.

Table 7. Weblog generation performance comparison

Algorithm	Statistical	RNN	GAN
Human score	0.4335	0.5400	0.6450
BLEU	0.5811	0.6482	0.7243

6 Conclusions

In this paper, we proposed a step forward towards automatic production of high-quality weblog using deep learning techniques, such as recurrent neural network and generative adversarial neural networks. We provided an analysis of state of the art, aimed to identify the techniques to be used in order to reproduce and improve the best performances reached today with generative approaches for discrete sequences of data. We first implemented the state of the art algorithm, that improves the performances reached with random techniques, using data mining and generating navigation sequences based on association rules. Then we implemented a recurrent neural network that tries to learn the probability distribution of the input data and is capable of predicting the right URLs to complete a given incomplete sequence with good performances when the number of features is not very large, while it is not robust in the other case. Finally, we developed the GAN by adding a convolutional neural network as the discriminator, to allow the RNN to improve itself, applying the so-called min-max game between the two networks. Our experiments support the hypothesis that generative adversarial neural networks are the best families of models to handle weblog generation and that they can outperform the recurrent models especially when the number of feature variables increases substantially. We showed that using both the BLEU and the Human metric, the GAN overcomes the RNN and the statistical approach when the generator is well trained. Instead, when the pre-train epochs for the generator are not enough or too much, the quality of the generated sequences is lower than that of RNN, but is still higher than the statistical one.

Future Work. In addition to the possibility of including more variables in the training of the network that could improve the quality of the generated weblog, we mentioned the work proposed by [3] for visualizing the statistics taken from weblogs on a graphical representation of a particular website or app, using a model-driven approach. With the GAN used, a future work could be to generate new weblogs and fed the model of the website with them. Then, one could compare two models where one of them is fed with human generated logs and the other with the GAN logs.

References

1. Basu, K.: Fake apache log generator (2015–2018). <https://github.com/kiritbasu/Fake-Apache-Log-Generator>
2. Berendt, B., Spiliopoulou, M.: Analysis of navigation behaviour in web sites integrating multiple information systems. *VLDB J. Int. J. Very Large Data Bases* **9**(1), 56–75 (2000)
3. Bernaschina, C., Brambilla, M., Koka, T., Mauri, A., Umuhoza, E.: Integrating modeling languages and web logs for enhanced user behavior analytics. In: Proceedings of the 26th International Conference on World Wide Web Companion, pp. 171–175. International World Wide Web Conferences Steering Committee (2017)
4. Denton, E.L., Chintala, S., Fergus, R., et al.: Deep generative image models using a Laplacian pyramid of adversarial networks. In: Advances in Neural Information Processing Systems, pp. 1486–1494 (2015)
5. Goodfellow, I., et al.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 2672–2680. Curran Associates, Inc. (2014). <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
6. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
7. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint [arXiv:1308.0850](https://arxiv.org/abs/1308.0850) (2013)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**, 1735–80 (1997). <https://doi.org/10.1162/neco.1997.9.8.1735>
9. Kwon, M.: Flog, an apache log generator (2017–2018). <https://github.com/mingrammer/flog>
10. Lin, C.H., Liu, J.C., Chen, C.R.: Access log generator for analyzing malicious website browsing behaviors. In: 2009 Fifth International Conference on Information Assurance and Security, pp. 126–129. IEEE (2009)
11. Mahoto, N., Memon, A., TEEVNO, M.: Extraction of web navigation patterns by means of sequential pattern mining. *Sindh Univ. Res. J.-SURJ (Sci. Ser.)* **48**(1), 201–208 (2016)
12. NASA: Nasa apache web log (1995). <ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html>
13. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: A method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL 2002, Stroudsburg, PA, USA, pp. 311–318. Association for Computational Linguistics (2002). <https://doi.org/10.3115/1073083.1073135>
14. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**(6088), 533 (1986)
15. Singh, N., Jain, A., Raw, R.S.: Comparison analysis of web usage mining using pattern recognition techniques. *Int. J. Data Min. Knowl. Manag. Process (IJDKP)* **3**, 137–147 (2013)
16. Vedaprakash, M.P., Prakash, M.P.O., Navaneethakrishnan, M.M.: Analyzing the user navigation pattern from weblogs using data pre-processing technique. *Int. J. Comput. Sci. Mob. Comput.* **5**, 90–99 (2016)
17. Yu, L., Zhang, W., Wang, J., Yu, Y.: SeqGAN: sequence generative adversarial nets with policy gradient. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)