

Université de Lorraine IDMC LORIA

MSc Natural Language Processing – 2018 - 2019
UE 705 – Supervised Project

Anomaly detection with deep learning models
Bibliography report

Students:
Esteban Marquer
Prerak Srivastava

Supervisors:
Christophe Cerisara
Samuel Cruz-Lara
Reviewer:
Denis Jouvet

January 12, 2019

Contents

Introduction

Logs are an important part of any computer ecosystem today but, to understand and make future decision on these logs is an hard and important task these days. Deep Learning has emerged as a key player to complete this motto due to its state of the art performance on various major tasks related to anomaly detection. Among those tasks, there is intrusion detection, denial of service (DoS) attack detection, hardware and software system failures detection, and malware detection. However, in recent years deep learning is often regarded as black box due to the need of understanding very hard mathematical complexity and algorithms, and the lack of direct and simple interpretability.

Our project centers on anomaly detection in large amounts of system logs, with very few occurrences of said anomalies in available data. The project thus concentrates on realizing a top notch predictive system, used to model the normal behavior of the logs. Deviations from that behavior can be considered anomalies.

Based on those observations, we try to explain in this work a major deep learning architecture, the DAN (Deep Averaging Network), which we are using in our project and is also a state of the art architecture for major Natural Language Processing (NLP) tasks. Apart from this work we also shed some light on other state of the art technique like RNN (Recurrent Neural Network) using attention mechanism, and LSTM (Long and Short Term Memory) Networks. We plan to demonstrate our model's performance and to illustrate its interpretability using the Los Alamos National Laboratory (LANL) cyber-security dataset in the upcoming parts. We will present that dataset along with our main dataset of industrial data from the PAPUD project in the last part of the report.

Chapter 1

Different artificial neural network architectures and techniques

Currently, in the literature we can find a variety of neural network architectures. By "architecture" we mean the major structure of a neural network (the way the neurons are organized in layers and how these layers are connected for example). We will present here some of those architecture that can be used for anomaly detection in system logs, along with techniques used along those models.

Also, as current literature shows increasingly [1], character-level models achieve similar performance than word-level models, while managing out-of-domain data way better (it is way less likely to encounter an unknown character than to encounter a new word).

1.1 Multi-layer Perceptron / Fully Connected Layers

The first model we ought to present is the multi-layer perceptron [6], a very simple neural network architecture. It is composed of multiple layers of neurons, with every neuron of a layer connected to all the neurons of the next layers. Between each pair of layers an activation function, also called non-linearity, is added.

Even if its basic concept dates of the 80s [13] and is far from being state of the art, it is still used in more complex architectures such as the Deep Averaging Neural Network.

1.2 Preprocessing

"Preprocessing" designates all the operations applied on the data to filter inappropriate data and adapt its representation to obtain the best quality data as input for the model.

For example, in [16] various preprocessing techniques are used on the input log lines data. In particular, they use a clustering technique on the raw text from multiple log sources to extract specific information from the logs and generate sequences that it be fed into a Long and Short Term Memory Network (LSTM, described subsection 1.5.1, page 3) network. By using such handcrafted feature extraction processes, they are able to improve the performance of their model by removing unnecessary information from the original data.

However, handcrafted feature extraction is costly to produce, and may remove potentially useful information.

In contrast to [16], our algorithm works almost directly on the logs, with the only preprocessing being the mapping of all the characters to numbers through an automatically generated dictionary. We also normalize the size of each log line, by padding them with a specific character or removing the excess characters. For reference, in [1] they need an additional step of

tokenization using know delimiters because they are also using words and not only characters.

1.3 Embeddings

Various deep learning NLP models use word or character embeddings [14, 6]. They are a kind of mapping that gives a way to represent efficiently the words (or character), instead of using 1-hot tensors (tensors of zeros, except for the cell corresponding to the word which is one). The mapping is either pre-trained or learned alongside the model. This way, a full dictionary can be represented with almost no loss in information in less than 200 dimensions (instead of one dimension per word). Quite the opposite in fact, as forcing a dimensionality reduction force the embedding to learn similarity between words.

1.4 Attention Models

Recently, a group of researchers have modified a LSTM (see subsection 1.5.1, page 3) and are now using it with attention mechanisms in order to capture long term dependencies on the system logs [1]. The attention mechanisms allows us to get some insight about what factors are affecting the model's decision, by observing which elements are selected by the attention weights. Attention mechanisms also enable to select recover information from a context, like previously seen elements, thus allowing the capture of long term dependencies. But, in contrast, in our model we did not use any kind of attention mechanism but rather kept it simple, because of the complexity of attention mechanisms and the high time and resource cost they induce. To get the long term dependencies between the log lines is currently not in the scope of our project, but is rater planned as way later step in the process to answer the problematic.

1.5 Order-sensitivity of the model

One last point of interest, that will directly lead us to the architecture we will use, is whether our model should be sensitive to the order of the elements in the input data or not. This divides the architectures in two families: unordered and syntactic. Unordered means that we convert the text to a bag of word embeddings, without a care for the order of the elements, while syntactic keeps order of the input text and is thus able to discover syntactic information.

For example with sentiment analysis, given a sentence like "The movie was entertaining but I don't like it very much" an unordered model like NBOW (see subsection 1.5.3, page 4) will have trouble dealing with the negation in "don't like", and thus will most probably return a positive label, which is a mistake. Syntactic aware models, like RNNs and LSTMs (see subsection 1.5.1, page 3), or CNNs (see subsection 1.5.2, page 4), are way more likely to handle such things.

1.5.1 Recurrent Neural Network (RNN) and Long and Short Term Memory Neural Network (LSTM)

RNN's architecture of is made in such a way that it can keep track of the phrase or the words that have appeared in the beginning of the sentence and thus can make final decision according to it. It has proven to be very efficient in many NLP applications [1, 9]. However, with this advantage also comes the disadvantage of having a large training time and hardware use. The main reason is that the non-linearities and matrix/tensor products at each node of the parse tree of a RNN are expensive, especially as model dimensionality increases. It also seems RNNs can sometime also struggle to deal with out of domain data.

LSTMs are a variant of RNNs, more complex and more powerful, but they also requires an even larger amount of training time and computations. They are considered one of the state of the art in the domain [12].

1.5.2 Convolutional Neural Network (CNN)

However, Y. Kim in his paper [11] shows that Convolutional Neural Network (CNN) can deal with both the dependency problem and the out of domain data, while giving effective results with low training time and hardware constraints. The CNN architecture has proven its efficiency with image processing, but it still proves to be very efficient with text [6, 5]. A CNN uses a sliding window (also called kernel) that run through the data to extract local information and process it. CNNs can have various channels with different kernel sizes, which give very good results because those various channels can capture different important information on different scales on a single given input sentence. For example, one channel can capture information about the word structure while another can capture grammatical information. However, the presence of multiple channels with various sliding windows, while composed of very simple operations, cause the computational cost to increases for large amounts of data (especially if we compare the training time with the one of a DAN (see subsection 1.5.4, page 4)).

1.5.3 Neural Bag of Word (NBOW)

The NBOW model just contains 4 major steps that are defined as follows [4]:

1. the conversion of text into tokens;
2. the conversion of tokens into embeddings;
3. the model average out the embedding which produces a vector "z";
4. it performs a softmax (operation that transform a set of values into a probability distribution) on "z" to get the probability of the labels.

During training, we apply the cross-entropy function to calculate the error and back-propagate the error throughout the model. Due to being composed of very simple functions (embedding, averaging, softmax), the NBOW model is quite simple and fast to train.

We have explained NBOW model here because it will serve as a baseline for DAN model because DAN is simply a modification of NBOW model.

1.5.4 Deep Averaging Neural Network

The Deep Averaging Network [7, 3] model works in 3 simple steps that are described as follows:

1. we take the vector average of the embeddings associated with an input sequence of tokens;
2. we pass that average through one or more feed-forward layers;
3. we perform a (linear) classification on the final layer representation (in our project, this step is replaced by the generation of the next log line).

An advantage of the DAN model is its high processing speed. Deep Averaging Network is a modification of an NBOW (Neural Bag Of Words Model), by adding the linear classification step to the model.

Choice of DAN

Although syntactic composition is very useful to obtain powerful models for example with LSTM, it requires a long training and expensive computations thus slows down the process. In our case, the dataset is very huge so such costs aren't really affordable. Even if CNN offer a less thorough syntactic composition support, they still require a larger amount of resources than DAN. However, with CNN the cost seem more affordable. Furthermore, DAN network can also be trained on various types of data without the need of large amount of pre-processing and also can be applied to the data with high syntactic variance.

Chapter 2

In-depth description of the Deep Averaging Network (DAN)

2.1 Internal structure of the DAN

As DAN is a modification of an NBOW model, with an additional perceptron between the average tensor and the softmax. It is a feed-forward neural network (it does not use recurrency nor convolutions), and each layer of the perceptron learns a more confined information of input sequence than the previous layer.

To be more concrete, take $s1$ as the sentence "I really loved Waffles performance in Belgium" and generate $s2$ and $s3$ by replacing "loved" with "liked" and then again by "did not like", then you will see that the embedding average of $s1$ and $s2$ is quite similar to each other, while the average for $s3$ is only a little bit different from the previous two. However, the meaning of $s3$ is really different from the other two. The hidden layers in the DAN are able to amplify those "small but meaningful differences in the word embedding average" [7], allowing it to achieve state of the art performance with very little computational cost.

The addition of 2 or 3 feed forward layers allows to increase the depth in the model and also allows to capture subtle variation in the input better than NBOW and computing each layer in feed forward layer is just a simple matrix multiplication. In practice the training time of DAN and NBOW is more or less the same.

So, instead of directly giving this output to the final layer we compute and give it to numerous layers in between as depicted in the Figure 2.1. The figure also shows the obvious scalability advantage of DAN over RNN, because the number of computations increase according to the size of the input data while the averaging in DAN absorbs the size variation.

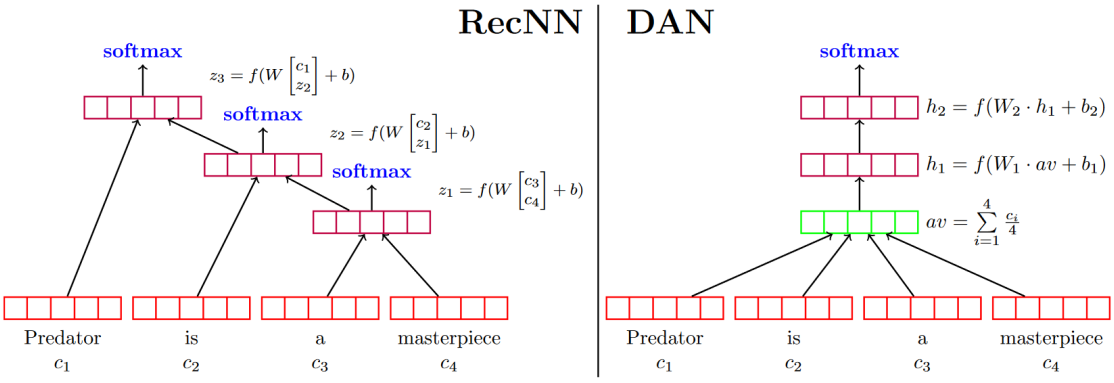

Figure 2.1: Comparison of the DAN model system with a basic RNN [7]

2.2 Improving Performances

To improve the performance of DAN network we have used we can perform dropout on input word sequences by randomly dropping a word in a sentence by a some defined probability. The basic motto of dropout is that it prevents overfitting of the data in the model, by randomly removing part of the data, thus artificially increasing the number of different examples. In DAN model we can drop word tokens or part of the embedding average. Using this method our input sequence to the model see different token sequence for each input X. However, dropout may also drop words or characters which are really necessary for the analysis of the input sequence. But most of time this method improves the performance of the model. We can note that the dropout technique work well with DAN models although it can alter and deteriorate the results if applied to other models like RNN or LSTM.

2.3 DAN For Predictive Maintenance

In our project we are using the DAN model to predict the next occurring system log in a sequence of many logs. As described in the upcoming sections, we have a very large data set (comparatively to the usual datasets used in deep learning). Compared with other state of the art architectures like the RNN or LSTM, which concentrate on achieving high performance with a low amount of data, the DAN would give us equivalent or better performance, while maintaining a very low training time necessary to draw the most of the amount of available data. DAN can also magnify small and meaningful differences from the different logs. Another interesting feature of, the DAN is its ability to incorporate out-of-domain data which is useful to maintain a stable predictive ability.

Chapter 3

Corpora used

3.1 The original corpora: the industrial dataset from the PAPUD project

The data from the PAPUD project is given to us by industrial partners of the project (BULL-ATOS). It is a set of system log files from their servers, which in total amount to more than 400GiB of compressed data; we are currently using a small subset of this data for our model.

The data from that dataset is confidential. Thus, the example we present in Figure 3.1 has been altered.

Timestamp	1524463200
Date	2018 Apr 23
Time	08:00:00
User	OOO
Process	authpriv
Message type	info
Message	access granted for user root (uid=0)
1524463200 2018 Apr 23 08:00:00 OOO authpriv info acce[...]ot (uid=0)	

Figure 3.1: Log line example from the industrial dataset (the full line has been abbreviated)

3.1.1 Preprocessing

As mentioned in section 1.2 (page 2), we try to do as little preprocessing on the data as possible. However, for the industrial data, our preprocessing (which was designed in a previous part of the project on the same data), is non-negligible.

The complete preprocessing is as follows:

1. we remove the Timestamp, Date, Time, and User elements, as they are really redundant and do not interest us for now;
2. we replace hexadecimal numbers/memory addresses by a specific character depending on the structure of the hexadecimal number;
3. we normalize all the lines to a length of 200 characters, by removing the excess characters and padding the shorter lines with a specific padding character;
4. we map of all the characters to numbers through an automatically generated dictionary.

3.2 Additional corpora: the LANL dataset

The other dataset we are using has be generated by Los Alamos National Library (LANL). This is a publicly available dataset [10], which contains around one billion log lines which is generated over the span of 58 consecutive days. The logs are about anonymized processes, network flow, DNS and authentication information [1]. Interleaved are attacks by their Red Team [1]. Our project work on the log lines which have the specific format which is the following:

Source user, Destination user, Source pc, Destination pc, Authentication type, Logon type, Authentication orientation, Success/failure

An example from the article [1] is presented in Figure 3.2.

Timestamp	1
Source user	C6@D1
Destination user	U7@D2
Source PC	C6
Destination PC	C6
Authentication type	Negotiate
Logon type	Batch
Autentication orientation	LogOn
Success/Failure	Success
1,C6@D1,U7@D2,C6,C6,Negotiate,Batch,LogOn,Success	

Figure 3.2: Log line example from the LANL dataset [1]

These log lines have been generated from desktop PC and active directory servers which are using Windows OS. In this dataset they have discarded all the lines that have machine listed as source user [1].

We use this dataset to test our model on intermediate- to large-size training dataset, with a simpler structure than the industrial dataset we got, with two objectives in mine: to use a simpler dataset to improve our model, and to compare the performance of our model on the LANL and industrial dataset.

3.2.1 Preprocessing

As mentioned in section 1.2 (page 2), we do very little preprocessing on the data from the LANL database, with only tree very simple process:

1. we remove the Timestamp field;
2. we normalize all the lines to a length of 126 characters, by removing the excess characters and padding the shorter lines with a specific padding character;
3. we map of all the characters to numbers through an automatically generated dictionary.

Note that in [1], they keep the timestamp information in the log line. However, we do not use them in our model as they do not interest us for now, and due to a consistency concern with the processing we do on the industrial dataset.

Conclusion

To conclude this report, we have justified our perspective on why we are focusing to use DAN architecture on both LANL dataset and PAPUD industrial dataset. First, we have tried to give a brief introduction about the different state of the art neural network architectures that can be used in our model, and have shown their major pros and cons for our project. Using this as a basis we explained the choice of DAN architecture before explaining its structure in more detail along with some potential ways to improve the performance of the DAN. We have also illustrated the datasets we are using on our model. You can find a description of our current two implementations of the DAN on these dataset in the annex part of the report, page 13. Hence, our main objective for the second half of the project would be to produce a very accurate line by line predictive model most likely based on DAN and to test it on our two datasets.

Annexes

Annex A: Current implementations

3 A first DAN implementation, towards 100% precision and overfitting, and processing large amounts of data

3.1 Description of the model

Layer	**Output**	
Input data (128 characters)	N	128 characters
Character dictionary	$N *$ dictionary size	128 * dictionary size
Embedding layer (output $E = 128$)	$N * E$	128 * 128 (embeddings)
Pooling layer (maxpool)	E	128 (max embedding)
Multilayer Perceptron (layer 1)	h_1	128
Multilayer Perceptron (layer 2)	h_2	128
Multilayer Perceptron (layer 3)	$N *$ dictionary size	128 * dictionary size
Softmax	$N *$ dictionary size	128 * dictionary size (probability distribution over the character dictionary)

Figure 3.3: Basic architecture of the DAN model from the PAPUD project

Between each layer of MLP we are using the ReLU activation function.

A simpler version of this model (with only the last layer of the MLP) was developed and used for the previous step of the PAPUD project. It has been tested and improved since the beginning of our project.

3.2 Current status

We have successfully used and updated the previous model on the industrial dataset and attained convergence (the loss is stable). However, the current model has a precision similar if not slightly lower than the previous simpler version (about 90% on the industrial dataset).

The current model's embedding is wrongly calibrated: we have about 64 different characters for the LANL dataset, less than 128 in the industrial dataset from the PAPUD project, and our embedding size is 128 which is way to large. This wrong parameter was overlooked in the previous development process and will be changed to a lower value of 16 or 32.

3.3 Future work

Our current target with this model is double: we want to achieve a near 100% precision, by all means, even if it means overfitting, and we want to train the model on a large dataset of 8GiB of compressed text data. The first goal is in good progress however the current training on the large dataset of 8GiB is abnormally slow. Improving the training time is thus our short time goal.

4 A second DAN implementation, evaluation on the LANL dataset, and testing a slightly different approach

4.1 Description of the model

Layer	**Output**	
Input data (100 characters)	$N = L * C$	100 lines * 100 characters
Character dictionary	$N *$ dictionary size	100 * 100 * 64
Embedding layer (output $E = 50$)	$N * E$	100 * 100 * 50 (embeddings)
Pooling layer (maxpool)	$N * 1 * E$	100 * 1 * 50 (per line max embedding)
Multilayer Perceptron (layer 1)	h_1	100 * 50
Multilayer Perceptron (layer 2)	h_2	100 * 50
Multilayer Perceptron (layer 3)	$C *$ dictionary size	100 * 64

Figure 3.4: Basic architecture of the DAN model for the LANL dataset

In the current model we are taking 100 consecutive lines and for each line the maximum character count is 100. So, In every line if the character count is less than 100 then we are implementing space padding to make the character count of 100. We choose the dimension of embedding as 50 so after that layer we have an output of size [100*100*50]. Between each layer of MLP we are using an activation function like ReLU. Our target with this model is a 101th line in the dataset that is what we will try to predict predicting. To compare the output of our model with the target output, we want to calculate the loss via cross entropy loss function. Because we want to predict the next log line, our target should be of size [100], corresponding to the characters in the 101th line in our dataset. Each value in target is between 0 and the dictionary size minus one, that value corresponding to the character's position in the character dictionary. Finally, the output of our final layer is of size [100*64].

4.2 Current status

As of now we have implemented the above architecture successfully to the dataset but we are yet to calculate the accuracy on the test set of the LANL data. Till now by using this model on the LANL dataset the loss is constantly decreasing (we have not reached convergence yet).

4.3 Future work

First of all our motive would be to perfectly apply DAN model on the LANL dataset and to get good accuracy on the test data and training data.

Bibliography

[1] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. *arXiv*, Mar 2018. URL: https://arxiv.org/abs/1803.04967, arXiv:1803.04967.

[2] Jason Brownlee. A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. *Machine Learning Mastery*, 07 2017. URL: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size.

[3] Xilun Chen, Yu Sun, Ben Athiwaratkun, Claire Cardie, and Kilian Weinberger. Adversarial Deep Averaging Networks for Cross-Lingual Sentiment Classification. *arXiv*, Jun 2016. URL: https://arxiv.org/abs/1606.01614, arXiv:1606.01614.

[4] Jocelyn D'Souza. An Introduction to Bag-of-Words in NLP. *Medium*, Jun 2018. URL: https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428.

[5] Vladimir Golovko, Mikhno Egor, Aliaksandr Brich, and Anatoliy Sachenko. A Shallow Convolutional Neural Network for Accurate Handwritten Digits Classification. *Communications in Computer and Information Science*, 673:77–85, Feb 2017. doi:10.1007/978-3-319-54220-1_8.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. [Online; accessed 11. Jan. 2019]. URL: http://www.deeplearningbook.org.

[7] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé Iii. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1:1681–1691, 2015. doi:10.3115/v1/P15-1162.

[8] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, 02 2016. URL: http://arxiv.org/abs/1602.02410, arXiv:1602.02410.

[9] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, 05 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness.

[10] Alexander D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, Jul 2017. URL: https://csr.lanl.gov/data/cyber1, doi:10.17021/1179829.

[11] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *arXiv*, Aug 2014. URL: https://arxiv.org/abs/1408.5882, arXiv:1408.5882.

[12] Christopher Olah. Understanding LSTM Networks, 08 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. MIT Press, Jan 1986. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

[14] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *CoRR*, 05 2018. URL: http://arxiv.org/abs/1805.09843, arXiv:1805.09843.

[15] Pavel Surmenok. Estimating an Optimal Learning Rate For a Deep Neural Network. *Towards Data Science*, 11 2017. URL: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0.

[16] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated IT system failure prediction: A deep learning approach. *undefined*, 2016. URL: https://www.semanticscholar.org/paper/Automated-IT-system-failure-prediction%3A-A-deep-Zhang-Xu/95193d2c016f1ee266b1dbf714678ce6bb1bb2ea.