Universiteit Leiden
The Netherlands

Opleiding Informatica

Anomaly Detection with

Deep Belief Networks

Jasper van Riet

Supervisors:
Wojtek Kowalczyk & Bas van Stein

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

03/08/2017

Abstract

When dealing with a large amount of data, using machine learning methods can often be appealing. When it concerns cybersecurity, that large volume of data is without a doubt present and so is an increasing amount of importance on safeguarding both public and private secrets. Problematic however, is the fact that machine learning ideally needs a labelled dataset that represent real-life conditions. When that is provided, in what is termed supervised learning, classifiers can achieve remarkable accuracy. Yet these labelled datasets are scarce. These datasets often contain highly sensitive data that first has to be anonymised before it can be released to the public. If it would be possible to have a similar accuracy without the need for a labelled dataset, that would be ideal.

This thesis focuses on this concept of unsupervised learning. In particular, it concentrates its efforts on the applicability and feasibility of deep belief networks. These networks can ideally detect anomalies by learning a distribution of the data. If an event is not consistent with the rest of the data, a deep belief network can highlight that. In particular, this thesis trains a model for every single user from the dataset, and then proceeds to determine whether an hour of activity is anomalous behaviour or not. The results show that the usage of a deep belief network is feasible from a technical perspective, with training the model only taking a number of seconds, yet it is hard to determine the accuracy when dealing with an unlabelled dataset, thus leading the author to conclude that more research is needed.

Contents

Chapter 1

Introduction

Due to the ever-increasing size and complexity of network infrastructure, and the increasingly visible impact on society of attacks on this very infrastructure, defending against attacks is becoming both harder and more important. Any system that is intended to help detect anomalous behaviour in this context requires both a high accuracy and a good ability to adapt to new threats, which are always in development. Detecting anomalies is important in order to detect and prevent unauthorised intruders or attacks. This is done via a so-called Intrusion Detection System, which can analyse log files that are routinely kept by server administrators. Log files generally keep track of most, if not all, important events on the network, from user logins to file transfers. Of course, keeping track of all these events means that the result is a very large amount of data.

The volume of data that is being dealt with is generally high enough that manual sorting through the data is not preferable, or even realistic. At the same time, intelligent decisions need to be made based on the data and the patterns within. Thus, a system is needed that is capable of handling the large amount of data yet is still able to provide high quality selections of cases to investigate. Machine learning techniques, particularly the quickly evolving deep architectures commonly termed deep learning, could potentially be a great solution here. A commonly used approach is via supervised learning techniques. This consists of training a network on a labelled dataset and training the network to classify input data into a certain class. Yet high quality labelled datasets for this purpose are hard to come by due to the large amount of work required (to label the data), in addition to the fact that such data is often confidential, which would require yet more man-hours to anonymise the data.

In 2015, [Ken15] published a data set representing 58 consecutive days of internal network data from the Los Alamos National Laboratory in the United States. The data is anonymised and it is intentionally not revealed when the data was captured. The data encompasses five different types: authentication data, records of user processes, DNS look ups, network flows and a small number of known red team events. In its entirety, the data contains 1,648,275,307 events. Since the red team data is of a particularly small size, the decision was made to handle this as an unsupervised learning problem: a problem where we do not have a training set to train the network on. Due to this, a Deep Belief Network, and its building block the Restricted Boltzmann

Machine, seem to be a good candidate for an effective system. Due to the fact that the Deep Belief Network will be used in a generative manner, it can be trained on the data and detect whenever a certain input vector is not consistent with the rest of the data: it is an anomaly.

This thesis is organised as follows: Chapter 2 discusses related work; Chapter 3 discusses concepts from the deep learning techniques used; Chapter 4 explains the methodology used; Chapter 5 evaluates the results and Chapter 6 concludes.

Chapter 2

Related Work

The detection of anomalies, or outliers, is an active field of research. Over the years the scientific community has provided a great number of solutions, ranging from statistical analysis to machine learning based solutions [CBK09]. With the advent of deep learning techniques in recent years, there is a great interest in whether these techniques can outperform currently used methods. Over the past years, a number of approaches have been investigated. A system based on a sparse auto-encoder in combination with softmax regression is proposed in [JNSA15]. Another approach used a Long Short Term Memory Recurrent Neural Network Classifier on the KDD CUP 1999 dataset (which is labelled) and was able to achieve a high accuracy [KKTK16]. Finally, yet another contribution delves into ways to protect trained deep networks from being tricked into missclassifying certain input vectors, also known as adversarial samples [PMJ+16].

Conversely, traditional methods can still hold their own. In [HRD16], the Dirichlet process was used to fit statistical Bayesian models to the data and was able to discover two source computers acting in bad faith in the Los Alamos data, which were verified via the red team data. Similarly, in-production systems have used signature based traffic identification since the early 2000s [VDMCCS00]. The issue with these systems however is that threats for which no signature exists are ignored and any changes at all in the network environment require manual labour to correct the system. Both of these issues mean that the ideal solution has yet to be found.

This specific research will explore the uses of deep learning, specifically the Deep Belief Network which consists of Restricted Boltzmann Machines, in anomaly detection. In 2013, the uses of a Discriminative Restricted Boltzmann Machine in a semi-supervised manner were explored in [FPCDS13]. In [BG14], a Restricted Boltzmann Machine was trained for malware detection and focused specifically on minimising false positives, getting very close to a rate of no false positives. However, the detection rate of malware did suffer as a result.

Chapter 3

Deep Learning

3.1 Restricted Boltzmann Machine

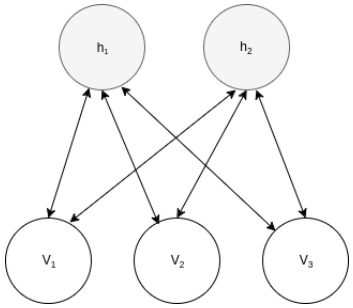

Figure 3.1: A Restricted Boltzmann Machine

A Restricted Boltzmann Machine (RBM) [Smo86, FH92, Hin02] (see Figure 3.1) is a type of stochastic neural network, consisting of two layers: a "visible" and a "hidden" layer. The visible nodes represent the input, which are connected to binary feature detectors which we call the hidden nodes. Together, they form a bipartite graph, which is what differentiates the Restricted Boltzmann Machine from the Boltzmann Machine. Connections are bidirectional. Neurons can have two states, activated by a probability. Each pairing of a visible and hidden node has an energy [Hop82] given by the following formula with v_i, h_j the binary states of visible unit i and hidden unit j, a_i and b_j their biases and w_{ij} the weight for the connection between them:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

With this energy, probability for every pairing can be calculated:

$$p(v,h) = \frac{e^{-E(v,h)}}{Z}$$

with Z a normalisation factor called the partition function:

$$Z = \sum_{v,h} e^{-E(v,h)}$$

The probability for a visible vector v is then retrieved by summing over all latent vectors:

$$p(v) = \frac{1}{Z} \sum_{h} e^{-E(v,h)}$$

This calculation is however intractable. Yet there is another way to calculate $p(v)$:

$$\sum_{h} e^{-E(v,h)} = e^{-F(v)}$$

With $F(v)$ the free energy of v:

$$F(v) = -\sum_{i} v_i a_i - \sum_{j} log(1 + e^{x_j})$$

Given these equations, the activation functions are as follows:

$$P(v_i = 1|h) = sigm(a_i + \sum_{j} h_j w_{ij})$$
$$P(h_j = 1|v) = sigm(b_j + \sum_{i} v_i w_{ij})$$

The result is a probability distribution, specifically a parametric model with parameters $\theta = (w, a, b)$, that models the joint distribution between the visible nodes (the inputs) and the hidden variables. Training a RBM means to attempt to maximize the likelihood of data coming from this distribution. Maximising the log-likelihood requires computing the gradient of log-likelihood function. This gradient has two phases, a so-called positive and negative one. In the positive phase the probability of the training data is increased by reducing the free energy while the negative phase decreases the probability of the generated samples. Problematic is that calculating the gradient is intractable, thus it has to be estimated. This is done via so-called contrastive divergence (CD). CD samples from the model in a certain amount of Gibbs sampling steps. Using Gibbs sampling as transition operator, samples of $p(v)$ can be obtained by running a Markov chain till convergence. CD does not wait till the chain has converged however, samples are obtained after just k-steps of Gibbs sampling. With the right setup, initialising the Markov chain with a training example instead of randomly, even $k = 1$ has been shown to suffice. [Hin10]

3.2 Deep Belief Networks

Deep Belief Networks (DBN) are models composed of multiple layers of RBMs stacked on top of one another, with the hidden layer of one RBM acting as visible layer for the next. [HS06] showed that these models can be trained greedily. First, the first layer is handled as a RBM to model the input with its visible layer. The hidden representation of this RBM is then used as the visible layer for the second layer, and repeat this step for the number of layers. Finally, the model is usually fine-tuned with some proxy for the log-likelihood or alternatively in a supervised manner. In our case, we are dealing with an unsupervised dataset and thus can't use the DBN in a supervised way.

In [SH07] and [KH11], deep autoencoders initialised by DBNs were used to map the input to binary codes, also known as semantic hashing. This paper uses a similar technique. Deep Belief Networks can be constructed in such a way that they represent an autoencoder. This is done by constructing a reverse network that reconstructs the input, just like an autoencoder. This reconstruction can then be compared to the original input data to see whether the input data fit the expectations of the model. The bigger the difference between the original input vector and its reconstructed counterpart the bigger the anomaly. We call this difference the reconstruction cost.

This difference is calculated by using the mean squared error (MSE). With X our input and Y our observed reconstructed output, both containing n elements, the MSE has the following formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - X_i)^2$$

Chapter 4

Methods

In almost any large network of users, usage patterns between different users will be anything but consistent. One user A will routinely connect to a large variety of machines during a standard working day, whereas another user B will arrive at work every day and log-in to the exact same computer, day after day. Keeping this in mind, the decision was made to train a network per user. This means that when user A connects to yet another computer that day, the network, in theory, knows that that is not necessarily anomalous behaviour, whereas when user B suddenly connects to a hundred different machines in an hour, alarm bells should be going off. Furthermore, features are extracted over the course of every hour of activity. In other words, a matrix is generated for every user with for every hour of activity (empty records are ignored) a collection of features that attempt to accurately describe the hour of activity.

A separate network is trained for every single user using (past) training data belonging to that specific user, and then applied to (simulated) future data. When applying to future data, we can extract statistics from the network, such as the reconstruction cost, that describe how well this hour of activity fits into the overall distribution of the user's behaviour. Cases with the biggest deviations from the norm are then investigated.

4.1 Data

The data published in [Ken15] contains several different types of network data. For this thesis, only the authentication data was used. The Los Alamos National Laboratory makes uses of the Windows Active Directory technology. The data contain 12,425 users, with around 6,000 more so-called computer users. The entirety of the data was de-identified. It is not specified in what timeframe the data was captured, only that the timespan is 58 (consecutive) days.

The authentication data contain authentication events which have been collected from Windows machines, servers and Active Directory servers. The data are in the following format:

```
1,C625$@DOM1,U147@DOM1,C625,C625,Negotiate,Batch,LogOn,Success
1,C653$@DOM1,SYSTEM@C653,C653,C653,Negotiate,Service,LogOn,Success
1,C660$@DOM1,SYSTEM@C660,C660,C660,Negotiate,Service,LogOn,Success
```

Each line contains an event, with the format being:

```
time,source user@domain,destination user@domain,source computer,destination computer,
authentication type,logon type,authentication orientation,success/failure
```

4.2 Features

Accurately describing an hour of activity for a user requires carefully crafted features. An extra degree of complexity is added due to training a separate network for every user: having too many features per user would result in a significant loss of performance, thus the amount of features has to be kept slim. See Table 4.1 for the features used.

Table 4.1: Features used

No.	Feature name	Description
1	attempted_logins	Number of attempted logins
2	unique_dest_users	Number of destination users connected to
3	unique_src_computers	Number of source computers used to connect
4	unique_dest_computers	Number of destination computers connected to
5	freq_dest_user	Frequency of the most common destination user
6	freq_src_computer	Frequency of the most common source computer
7	freq_dest_computer	Frequency of the most common destination computer
8	perc_same_src_as_dest_comp	Percentage of connections with same source and destination computer
9	perc_successful_logins	Percentage of logins that is successful

4.3 Preprocessing

The data was processed on a per user basis. The data is first split up into two sets, a training set and a test set, 80% and 20% of the data for that user respectively, which will be used in the experiments. This is done chronologically, so the test set represents the last 20% of the records. Then, the data has to be scaled. All columns from the training set are normalised to an interval between 0 and 1. This is done using the following formula:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

with x the original data and x' the normalised data. This is not just done for feature scaling purposes. While a RBM is expecting binary data, this is a way to "cheat" and still input real-valued data, by inputting the data as probabilities. Important to note here is that $max(x)$ and $min(x)$ are saved in memory. The test set is then

normalised with the same formula, but with the $max(x)$ and $min(x)$ from the training set, to simulate real-life conditions. Then, the data indices for the training set are shuffled randomly, while the test set is kept intact. This is yet again done in order to simulate real-life use of such a system, where the model will be trained once and then used to evaluate incoming logs.

4.4 Running experiments

Once preprocessing is done, the data is fed into the network. Multiple types of experiments have been run on the data. First, the data was explored using just a single RBM. This is done by first training the RBM on the training set and then using the trained model to evaluate entries from the test set. For any particular test entry, the free energy is calculated and compared to the average free energy for the training set. If the free energy is particularly large (and thus inconsistent with the rest of the data), this input vector is deemed an anomaly. The RBM has 5 hidden nodes and is trained with a learning rate of $0,01$ in 60 epochs, which is the maximum amount of epochs that the RBM can be trained before it starts overfitting. This was determined by calculating the free energy gap between the same entries from the training and the test set every epoch. When this gap starts growing, the RBM is overfitting, see [Hin10]. The number of nodes and the learning rate was determined using advice given in [Hin10], in addition to comparisons of the free energy.

Experiments for the autoencoder DBN are a similar story. Pretraining is performed solely on the training set. Come the finetuning stage, we carefully monitor whether the network is overfitting. This is done by going through the entire training test set every epoch and calculating the mean reconstruction cost. If the reconstruction cost keeps improving, finetuning is allowed to keep going. If not, finetuning is stopped early, since in that case the network is overfitting. This is called early-stopping. Once the network is trained for a user, we can use it to evaluate any data we input. In this case we use the test set. Every entry is input into the network, after which the reconstruction cost is retrieved. When the reconstruction cost for a certain entry is significantly higher than it should be, that entry is looked at further. Thanks to keeping track of the original data in the preprocessing phase, the original data for that hour of activity can be pulled up, in addition to the way in which the input vector differs from what the network expected. This can then be studied by a domain expert.

For the autoencoder DBN, just a single hidden layer was used. This layer has 9 nodes. The number of nodes was determined by performing experiments and comparing the reconstruction costs. Just a single layer was used because reconstruction costs went up with more layers. The learning rate was set to $0,01$. For the pretraining phase, 100 epochs were run while the finetuning phase does 1000 epochs, if it is not stopped early.

Both of these are adaptations of the code written for [dlt], using Python with the Theano library. The RBM was used as-is, whereas the DBN given in [dlt] was changed to an autoencoding DBN.

4.5 Evaluation

For evaluating the performance, a number of approaches are used. The first is based on the reconstruction cost, where a higher reconstruction cost equals more anomalous, the second on the interquartile range. The interquartile range (IQR) of a distribution is the difference between the 75th ($Q3$) and the 25th ($Q1$) percentiles. In other words:

$$IQR = Q3 - Q1$$

Having found the IQR, we can now use it to find outliers in our distribution. An outlier is defined as any value that falls below $Q1 - 1.5 * IQR$ or any value above $Q3 + 1.5 * IQR$. We can calculate whether the maximum observation in a distribution is an outlier by transforming these into the following formula:

$$max(x) = Q3 + \alpha * IQR$$

We can then collect the α for every observation. If $\alpha > 1.5$, we can conclude that this observation is an outlier.

Chapter 5

Results

This section describes the results of the performed experiments. All experiments were run on a single thread on a machine with 16 Intel Xeon E5-2630v3 CPUs running at 2.40GHz (32 threads). For the RBM, training on a single user took 10.2 seconds on average. See Figure 5.1 for the free energies produced for two users. While results vary among users, generally the theme is similar: most values are in a certain range with a few extremes here and there. Results are highly influenced by the amount of data available for a certain user, as can be expected.

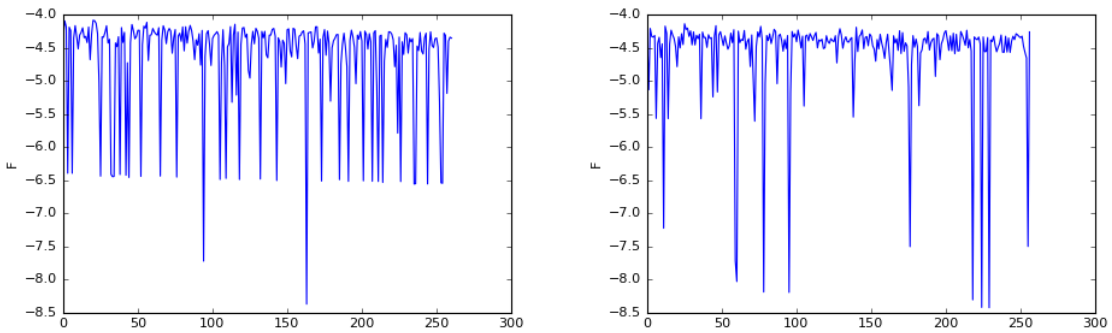

Figure 5.1: Free energies of the test sets of two users

The DBN runs faster, it takes only 3.9 seconds to do both pre-training and finetuning per user. Running over the entire dataset takes 13 hours, 48 minutes and 55 seconds.

During that time every user is evaluated using the methods described in Chapter 4. For every user, the interquartile range is calculated for the distribution of reconstruction. From there, the alphas are calculated. The result can be seen in Figure 5.2. Again, when $\alpha > 1.5$, this is deemed an outlier. As can be seen, there are many users that fit those criteria. It is unlikely that there are this many users with anomalous behaviour. It seems that this is an approach that doesn't work.

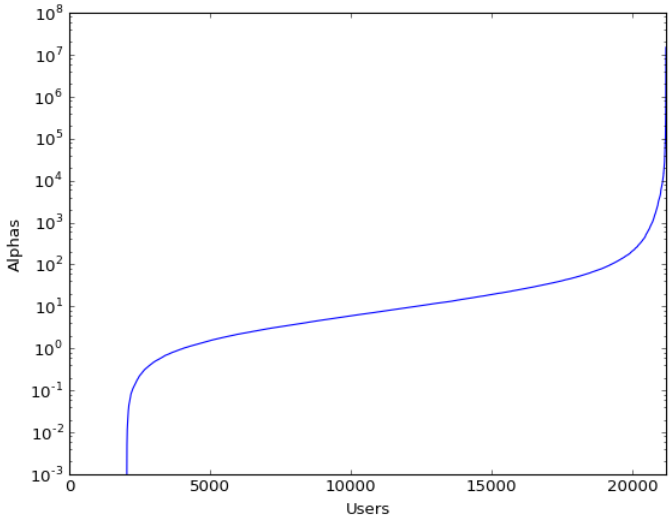

Figure 5.2: Interquartile range alphas for each user

The mean reconstruction cost for the test set for each user can be seen in Figure 5.3. It seems clear that these provide a lot more consistent results than the interquartile range. A zoomed-in plot of the peak in Figure 5.3 can be seen in Figure 5.4. To examine the workings of the network more closely, we will now take a look at some of the users who figure as the biggest peaks in Figure 5.3. It should be noted here that Figure 5.3 does include Windows Active Directory computer accounts, which depending on the set-up of the network may or may not be a person and do have different behaviour from the users. See Table 5.1 for the first example of such a peak, user `U7535`. For this user in this hour of activity, the values that fall out of the line of expectation are `unique_dest_users`, `unique_src_computers` and `perc_same_src_as_dest_comp`, which are all higher than expected by the network and finally `perc_successful_logins`, which is of significantly lower value than is expected for this user.

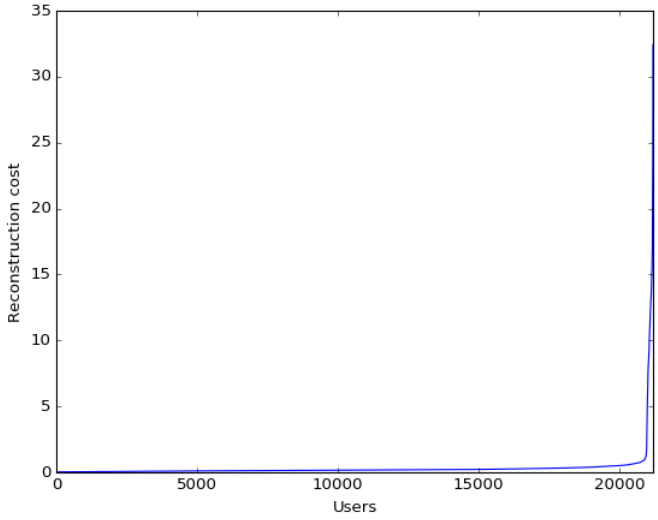

Figure 5.3: Mean reconstruction cost by DBN for each user

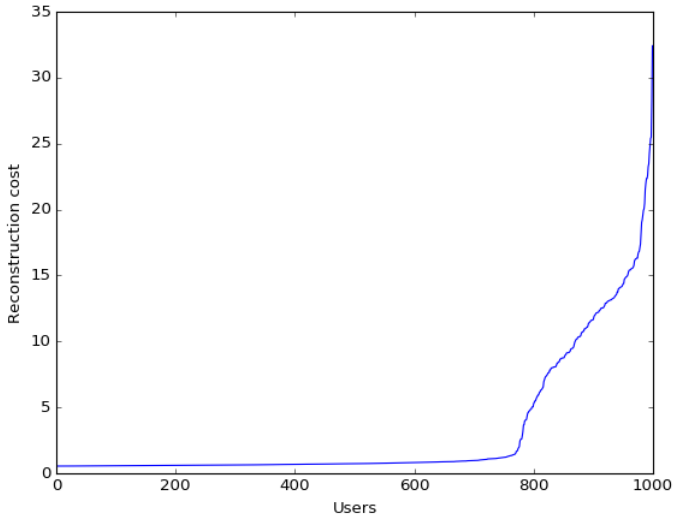

Figure 5.4: Mean reconstruction cost by DBN for the 1000 users with the highest reconstruction costs

Another user to look at would be `U10800`. Of the 30 highest peaks, this user represents 3. The interesting thing here is that while the reconstruction cost is very large for all three, they do not immediately seem like large anomalies. Yet in the context of this user, the DBN says that these vectors representing an hour of activity are nothing like the normal behaviour of `U10800`. In the case of the example shown in Table 5.1, most values are nothing too out of the ordinary, with one exception: `perc_successful_logins`. This percentage is much lower than expected. The other peaks for this user are very similar: most values are as expected, with `perc_successful_logins` having a low value.

Table 5.1: Hour 1287 (day 53) for user `U7535` (left), hour 1306 (day 54) for user `U10800` (right)

No.	Feature name	Value
1	attempted_logins	19
2	unique_dest_users	1
3	unique_src_computers	2
4	unique_dest_computers	2
5	freq_dest_user	19
6	freq_src_computer	10
7	freq_dest_computer	10
8	perc_same_src_as_dest_comp	1.0
9	perc_successful_logins	0.0

No.	Feature name	Value
1	attempted_logins	13
2	unique_dest_users	1
3	unique_src_computers	2
4	unique_dest_computers	3
5	freq_dest_user	13
6	freq_src_computer	9
7	freq_dest_computer	7
8	perc_same_src_as_dest_comp	0.846
9	perc_successful_logins	0.308

We can further illustrate that these cases of a low percentage of successful logins for user `U10800` is anomalous by generating boxplots for the (scaled) training set and test set respectively, see Figure 5.5. As can be seen, there is not a single example in the training set where the percentage of successful logins is as low as values that can be found in the test set.

Table 5.2: Hour 802 (day 33) for user U1597

No.	Feature name	Value
1	attempted_logins	7
2	unique_dest_users	1
3	unique_src_computers	2
4	unique_dest_computers	2
5	freq_dest_user	7
6	freq_src_computer	5
7	freq_dest_computer	5
8	perc_same_src_as_dest_comp	1.0
9	perc_successful_logins	1.0

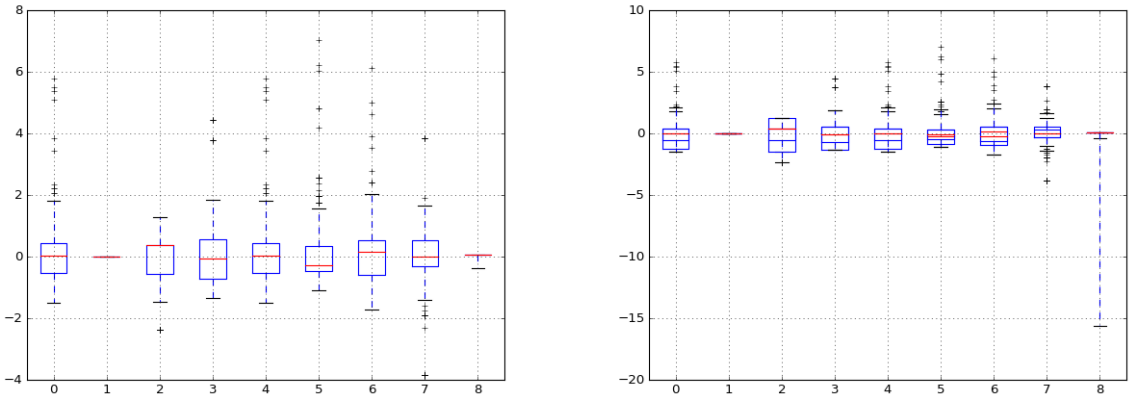

Figure 5.5: Distribution of values of features for the training set (left) and test set (right) for user U10800

Now for an example showing the limitations of the network, see Table 5.2. In this case, `attempted_logins`, `freq_src_computer` and `freq_dest_computer` are higher than expected. Yet upon manual inspection, it is hard to classify this as an anomaly. Perhaps in the context of the user this behaviour is different than normal, say, with more logins than usual, but it seems highly likely that this case is innocent. It is here that we start seeing fundamental limitations in the features that were chosen. With a different set of features this case might not have been picked out at all.

Chapter 6

Conclusions

This paper looked into the feasibility of anomaly detection with deep belief networks, in particular with regards to unsupervised learning. We were interested in both the technical feasibility, e.g. what is the performance like, and its abilities to accurately detect anomalies. This was done on a dataset that is entirely unlabelled.

On the first front, performance, we get to conclude that using DBNs for anomaly detection is technically feasible. With just a number of seconds needed to train a network with 9 features for a user, it is entirely realistic to use such a network in real life conditions. When it comes to accuracy in detecting anomalies, it becomes a much harder question to answer. This is in large part due to the fact that the dataset is unlabelled and thus we do not know whether what the network sees as anomalies are in fact anomalies. This simple fact means we have a hard time training the network and finding the best way to define an anomaly. The most that can be done in this regard is to pick out a number of examples and have a domain expert study them. From our studying of the data there seem to be a number of actual anomalies that were detected by the network, alongside a number of cases of what are seemingly false positives.

There is a lot that can be done to improve on this research. Since performance has been established to be satisfactory, a large focus should be put on the classifying accuracy; just how well can DBNs find anomalies? In doing this, benefits can be found in using a different dataset and taking the time to devise better features that are more representative of a user's activity. With the right features, it is likely possible to have a DBN with more layers, thus allowing the network to create more complex connections between those same features, which in theory would be beneficial.

Bibliography

[BG14] Răzvan Benchea and Dragoş Teodor Gavriluţ. Combining restricted boltzmann machine and one side perceptron for malware detection. In *International Conference on Conceptual Structures*, pages 93–103. Springer, 2014.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[dlt] Deep learning tutorials. `http://web.archive.org/web/20170614015723/http://deeplearning.net/tutorial/`. Accessed: 2017-06-14.

[FH92] Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in neural information processing systems*, pages 912–919, 1992.

[FPCDS13] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomputing*, 122:13–23, 2013.

[Hin02] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[Hin10] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[HRD16] Nick Heard and Patrick Rubin-Delanchy. Network-wide anomaly detection via the dirichlet process. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*, pages 220–224. IEEE, 2016.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[JNSA15] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on*

Bio-inspired Information and Communications Technologies (formerly BIONETICS), New York, NY, USA, volume 35, page 2126, 2015.

[Ken15] Alexander D. Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, June 2015.

[KH11] Alex Krizhevsky and Geoffrey E Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

[KKTK16] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *Platform Technology and Service (PlatCon), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[PMJ+16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[SH07] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *RBM*, 500(3):500, 2007.

[Smo86] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[VDMCCS00] Jacobus Van Der Merwe, Ramon Caceres, Yang-hua Chu, and Cormac Sreenan. mmdump: A tool for monitoring internet multimedia traffic. *ACM SIGCOMM Computer Communication Review*, 30(5):48–59, 2000.