Self-supervised Pre-training on LSTM and Transformer models for Network Intrusion Detection

Optional Subtitle of the Thesis

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Embedded Systems

by

Jonas Ferdigg, BSc
Registration Number 01226597

to the Faculty of Electrical Engineering and Information Technology

at the TU Wien

Advisor: Univ. Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby
Assistance: Univ.Ass. Dott.mag. Maximilian Bachl

Vienna, 1st January, 2001

Erklärung zur Verfassung der Arbeit

Jonas Ferdigg, BSc

Hiermit erklÃďre ich, dass die vorliegende Arbeit gemÃďÃ§ dem Code of Conduct âĂŞ Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulÃďssige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt Ãijbernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im InâĂŞ noch im Ausland in gleicher oder in Ãďhnlicher Form in anderen PrÃijfungsverfahren vorgelegt.

Wien, 1. Jänner 2001

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Contents

CHAPTER 1

Introduction

1.1 Motivation

With the progressing digitalization of evermore aspects of society, cyber security will always be a relevant issue as no system will ever be fully secure. Preventing possible cyber attacks by developing more robust systems is one way to mitigate the issue, the other is preventing already existing faults from being exploited as not every vulnerability can be patched easily as it is the case with e.g. DoS and bruteforce attacks. To stop such attacks it is necessary to identify them within the vast flow of ordinary network traffic which gives rise to the need of Intrusion Detection Systems (IDS). State-of-the-art IDSs apply two methods to detect occurring attacks: Signature-based detection and statistical anomaly-based detection. Signature-based detection looks for known patterns or signatures within packets and data streams to identify incoming attacks . Statistical anomaly-based detection focuses on differentiating between normal and abnormal behavior in the system and raises an alert if the latter is identified. The problem with signature-based detection is that unknown attacks are ignored and anomaly-based detection is still not sufficiently accurate and prone to false positives . The rise of Machine Learning (ML) gave opportunity to use the mighty pattern recognition capabilities of Neural Networks (NNs) for intrusion detection. As ML is a rapidly developing field its steady improvement fueled the advance of NN based IDSs which start to show promising results . NNs however are still mostly trained in a supervised fashion, namely by providing labeled examples of cyber attacks for the NN to learn from. This again poses the problem, that only known attacks can be identified, but new attacks that are sufficiently similar to old attacks can also be identified, which is not the case with mere signature-based detection. As with every form of supervised training on NNs, labeled data is harder to come by while unlabeled data is often abundant and certainly so for network traffic data. For this reason, self-supervised training/pretraining is seeing increased use in the realm of ML , as unlabeled data can be used to boost the performance without the need

insert reference to state of the art ids

give examples for IDSs lacking accuracy

give examples for NN based IDSs

give examples of self supervised machine learning

for expensive labeled data. One of the most noteworthy examples of the effectiveness of self-supervised pre-training for Neural Networks in the realm of Natural Language Processing (NLP) is Bidirectional Encoder Representations from Transformers (BERT) [DCLT18] developed by Jacob Devlin *et alteri* from Google AI Language. BERT is based on the state-of-the-art Transformer architecture [VSP+17a] and uses a series of proxy tasks like word masking and next sentence prediction to teach the network about syntax and grammar in a self-supervised fashion. The pre-trained network can then be fine-tuned for more specific tasks like question answering or text classification. Analogous, it would be highly beneficial if these or similar pre-training mechanisms could be used to bolster performance of ML based IDSs by improving the classification of network flows, at the most basic level, into cyber attack vs. no cyber attack.

As the technologies mentioned above are fairly recent (Transformers Dec 2017, BERT May 2019) and the design space for solutions in the context of ML for cyber security is substantial, there has not yet been sufficient inquiry into the possibilities of these new methods when applied to the problems posed by Intrusion Detection and cyber attack classification. NN performance also improves with the steadily increasing capabilities of modern Graphics Processing Units (GPU) which makes this a promising concept that can be improved upon by future more powerful hardware.

1.2 Research Questions

In this thesis we inspect if the flow classification performance of Long Short-Term Memorys (LSTMs) and Transformer-Encoder networks can be improved with self-supervised pre-training in a scenario where only little labeled and a lot of labeled data is available. In our context this means a ratio of 1:1000 for labeled to unlabeled data. For performance we are mainly looking at the accuracy of classification, but we are also keeping track of the False Alarm Rate (FAR). The problem to solve is a binary classification problem for which the model is to group flows into *attack* and *no-attack*.

- R1: Can self-supervised pre-training improve the flow classification capabilities of an LSTM model?
- R2: Can self-supervised pre-training improve the flow classification capabilities of a Transformer-Encoder model?
- R3: Which pre-training tasks improve accuracy and which do not?

1.3 Approach

To answer these questions we conduct a series of experiments. In these experiments we devised different proxy tasks for the model to solve in a self-supervised fashion. Solving these proxy tasks serves as pre-training for the network during which it learns the structure of the data and to form abstract representations within its latent space. After

the pre-training we train the network with very little labeled training data to teach it how it should classify the flows. These experiments show if pre-training can improve accuracy of the model when compared to only training it with the same amount of labeled data but no pre-training. They also show which pre-training methods are more and which are less beneficial for classification accuracy.

1.4 Contribution

- Implementation of a pre-trainable LSTM model and training suite
- Implementation of a pre-trainable Transformer-Encoder model and training suite
- Inquiry into the benefits of pre-training for sequence-to-sequence models in the context of Network Intrusion Detection Systems (NIDSs)
- Development of new pre-training methods for LSTMs and TransformerEncoder models in the context of NIDSs

Here provide a list of the contributions of your work.

Suggestion (especially for dissertations): provide a table with research questions, methods used to answer each, and major findings and the section in which to find details.

1.5 Structure

After this introduction section we will provide some background information and define terminology used throughout the thesis 2. Subsequently we provide an overview of the current state-of-the-art of NNs for sequence-to-sequence modeling 3, pre-training for such models and ML supported NIDSs in general. Reasoning behind our methodology, and other decisions made, can be found in its dedicated section 4. A detailed description of the conducted experiments can be found in the section *Experiments* 5 with the goal to make them as reproducible as possible. A structured comprehension of experiments conducted is provided in the section *Results* 6. Finally, in the sections *Discussion* 7 and *Conclusion* 8 we discuss successes and failures and draw conclusions from our findings, including pointers for future research.

CHAPTER 2

Background

Artificial Neural Networks (ANNs) have shown great improvements over the last years due to increasing compute power, more sophisticated models and smarter training algorithms . ML and ANNs have long found their way into many commercial applications and many scientific fields have successfully applied this relatively new method of data processing to further their own research. It was only logical that researchers and companies have also started to look into the possible benefits this emerging technology could have for Network Security applications . ANNs are especially suited for IDSs due to their capability to classify data with high accuracy. To harness the power of ML for the purpose of Network Security, we made use of existing methods and models which we will summarize in this section.

cite papers

cite papers

cite papers

2.0.1 Machine Learning

2.0.2 Artificial Neural Networks

Named after their resemblance to neurons in a brain, ANNs are systems comprised of connected nodes called *artificial neurons*. Analogous to synapses, nodes communicate *via* connections called *edges* by sending "signals" to other nodes. Signals are represented as scalar real numbers. The output signal from a sending node is multiplied by the weight of the edge the signal is "traveling" on. Each node calculates its output signal by applying a non-linear function to the sum of its input signals. Signals travel forward through the network from the first to the last layer, but usually not within layers. There are various types of ANNs like Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs) which have many derivations themselves but they all operate on the before stated principal of signals traveling through the network which get transformed at each node by a differentiable non-linear function. The most popular non-linear function at this time is the Rectified Linear Unit (ReLU) function. Without training an ANN performs an input transformation that depends on the initialization values of its weights, often called

parameters. The network is trained to perform a desired transformation by adjusting its weights/parameters through virtue of *back-propagation*. The network produces output $\hat{y}$ at the last layer after processing input x. A scalar cost/loss value is calculated by a *loss function* $C(\hat{y}, y)$ as a measure of difference between the networks output $\hat{y}$ and the target output y. For classification tasks the loss function is usually cross entropy loss and for regression Squared Error Loss (SEL) is typically used. Back-propagation computes the gradient of every weight in the network with respect to the loss function by applying the chain-rule for every layer down to every weight. After calculating the gradient for every weight, a gradient method like Stochastic Gradient Descent (SGD) is used to iteratively update all weights in order to minimize $C(\hat{y}, y)$.

reference cross entropy loss

find/create graphic

2.0.3 Recurrent Neural Networks

The broader concept behind all RNNs is a cyclic connection which enables the RNN to update its state based on past states and current input data [YSHZ19]. Typically, an RNN consists of standard tanh nodes with corresponding weights. There are different kinds of RNNs like continuous-time and discrete-time or finite impulse and infinite impulse RNNs. Here we will only look at discrete-time, finite impulse RNNs as we will only be using those. This type of network, e.g. the Elman network [Elm90], is capable of processing sequences of variable length by compressing the information from the whole sequence into the *hidden layer*. The model produces one output token for each input token, so the transformation is sequence-to-sequence where input and output sequences are of equal length. One input sequence consists of a sequence of real valued vectors $x^{(t)} = x^{(1)}, x^{(2)}, ..., x^{(T)}$ where T is the sequence length. From this input sequence, an output sequence of real valued vectors $\hat{y}^{(t)} = \hat{y}^{(1)}, \hat{y}^{(2)}, ..., \hat{y}^{(T)}$ is produced. To train an RNN pairs of input and target sequences $(x^{(t)}, y^{(t)})$ are provided from which, analogous to the training of ANNs in general2.0.2, a differentiable loss function $C(\hat{y}^{(t)}, y^{(t)})$ can be calculated which can again be minimized by applying back-propagation. In theory, RNNs can process data sequences of arbitrary length, but the longer the sequence, the deeper the network gets i.e. the longer the gradient paths. This leads to complications when relevant tokens are further apart in the sequence as the RNN is not capable of handling such "long-term dependencies" [YSHZ19]. Long gradient paths in RNNs might also cause the gradient to become either very small or very large, which results in the known *vanishing gradient* or *exploding gradient* problems correspondingly and cause training to either stagnate or diverge. The LSTM improves upon RNNs by making the gradient more stable and allowing long-term dependencies to be considered in the learning process.

give a more formal description of RNNs

find/create graphic

2.0.4 Long Short-Term Memory

Introduced by Hochreiter and Schmidhuber in 1997 [HS97], the LSTM model mitigates the vanishing and exploding gradient problem by replacing the tanh nodes in the hidden layer of a conventional RNN with *memory cells* as seen in 2.1. A memory cell is comprised of *input node*, *input gate*, *internal state*, *forget gate* and *output gate*.

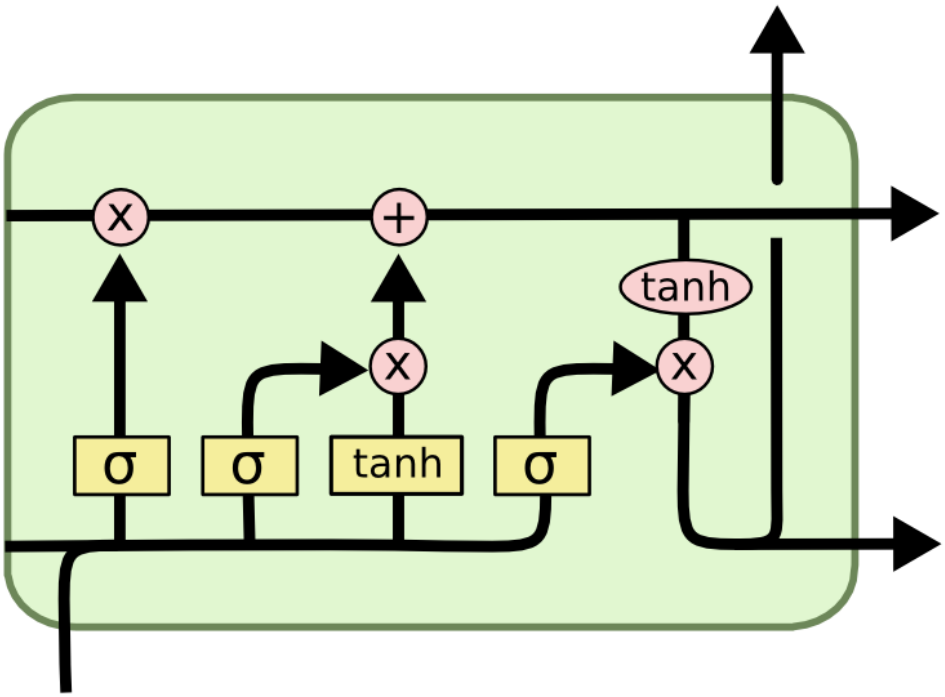


Figure 2.1: One LSTM memory cell [Lip15]

In contrast to an ordinary RNN, an LSTM has two memory states: the hidden state $h^{(t)}$ and the *cell state* $C^{(t)}$. Three gates enable the cell to control the flow of information and its effects on the cell state. For this purpose, gates in an LSTM consist of a point-wise multiplication with a vector that holds values between 0 and 1. The three sigma activations seen in 2.1 produce the gate vectors. The input gate $i^{(t)} = \sigma(W^i[h^{(t-1)}, x^{(t)}] + b^i)$ controls whether the memory cell is updated. The forget gate $f^{(t)} = \sigma(W^f[h^{(t-1)}, x^{(t)}] + b^f)$ controls how much of the old state is to be forgotten. The output gate $o^{(t)} = \sigma(W^o[h^{(t-1)}, x^{(t)}] + b^o)$ controls whether the current cell state is made visible. The weight matrices W^i, W^j and W^o decide how information is processed by the cell and are learned parameters. The cell state is updated by addition with the vector $\bar{C} = \tanh(W^C[h^{(t-1)}, x^t] + b^C)$ after multiplication with the input gate vector $i^{(t)}$. The repeated addition of a tanh activation distributes gradients and vanishing/exploding gradients are mitigated.

2.0.5 Adam Optimizer

2.0.6 Attention and Transformers

2017 Vaswani et al. published a paper with the ominous title "Attention is All you Need" [VSP+17b], referring to the already known attention mechanism which is used to model dependencies within a data sequence over longer distances. The authors proposed the Transformer model consisting entirely of self attention mechanisms to model sequences and therefore diverge from the recurrent architectures of RNNs and LSTMs. Attention is a mechanism to capture contextual relations between tokens in a sequence, e.g. words in a sentence. For every token in the input sequence, an attention vector is generated which represents how relevant other tokens in the input sequence are to the token in question. While attention can be implemented in different ways, the authors chose the scaled dot-product attention defined as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

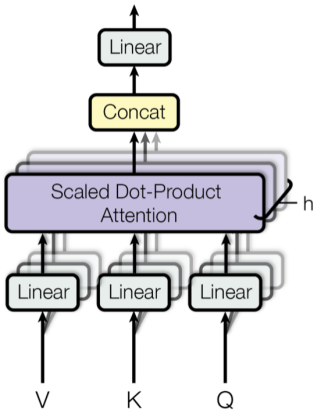


Figure 2.2: Self attention layer of Transformer by [VSP+17b]

"An attention function can be described as mapping a query and a set of key-value pairs to an output" [VSP+17b]. Q, K and V are matrices composed of query, key and value vectors for every token with respect to every other token in the sequence. Vaswani et al. proposed the use of Multi-Head Attention mechanism suggesting the use of multiple independent attention heads which are generated by linear projection of the original Q, K and V matrices by different learned matrices W_i^Q, W_i^K and W_i^V for $i = 1, ..., h$ where h is the number of desired attention heads. The attention vectors of the different attention heads are again concatenated and projected by matrix W^Z again resulting in a single combined attention vector instead of h vectors. This results in the formulation

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), i = 1, ..., h \tag{2.2}$$

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.3}$$

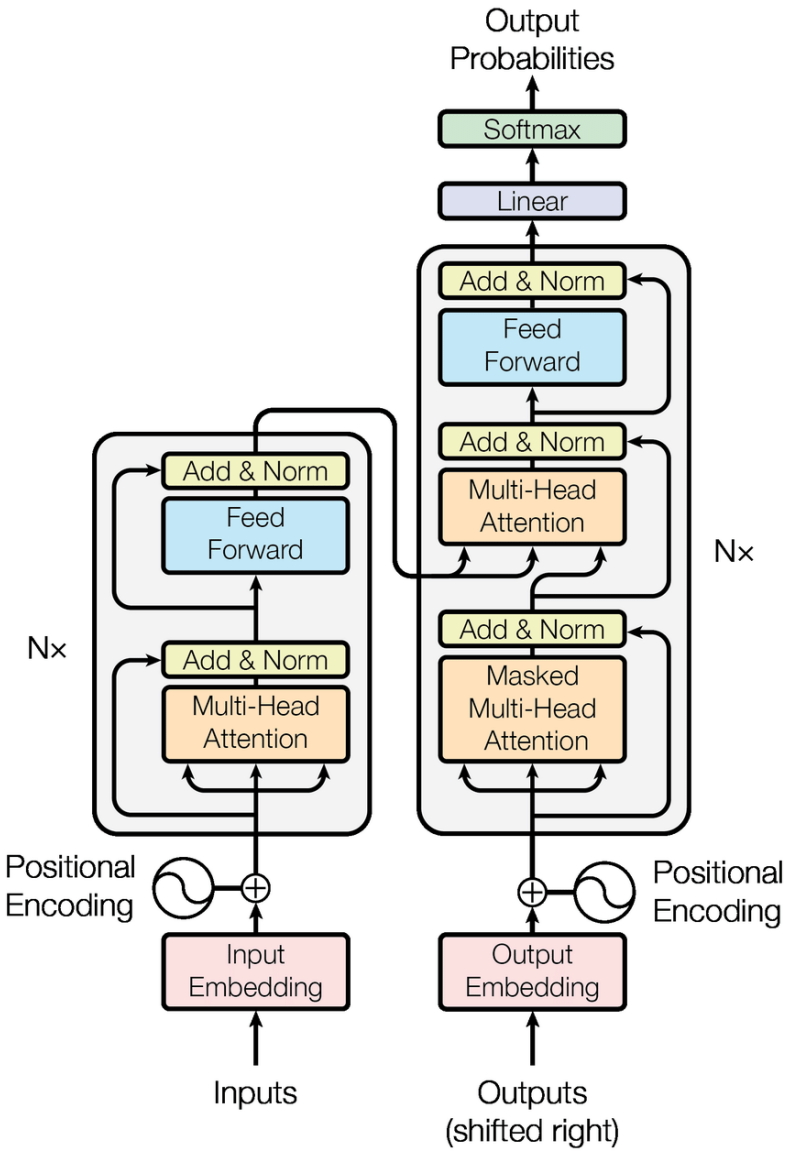


Figure 2.3: Complete Transformer Encoder-Decoder Model as proposed by [VSP+17b]

2.0.7 Self-supervised Learning

2.0.8 Auto Encoder

2.0.9 Pre-Training and Fine Tuning

2.0.10 Terminology

In addition: Abbreviations and mathematical notation should be put in a list in the beginning of the thesis

CHAPTER 3

State of the art

Here provide an overview of the related state of art. Look for papers that are closest to the research you are doing Suggestion: make a table with the related papers and compare them wrt to different criteria, for instance

- Findings: What do they claim (main findings)
- Data: What data set they are using
- Methods: Which methods did they use?
- Reproducibility: Is it possible to reproduce the results? (e.g., is the data available? are all parameter settings provided? Is source code provided?)
- Relevance (How relevant is it for your work)

In the last paragraph explain how your work differs from the existing works.

CHAPTER 4

Methodology

- explain why these experiments are used
- explain metric for comparing results (accuracy, false alarm rate)
- short summary of code?

Here describe the methodology you use and why you decided to use it. e.g., theoretical considerations, simualtons, experiments, measurements, testbeds, emulations, etc. What concepts are used.

Also explain which metrics you use to measure success or failure (e.g., detection performance with accuracy, recall, precision, f1 score, RocAUC, etc.)

Provide a figure (see example figure 4.1) to describe the processing steps

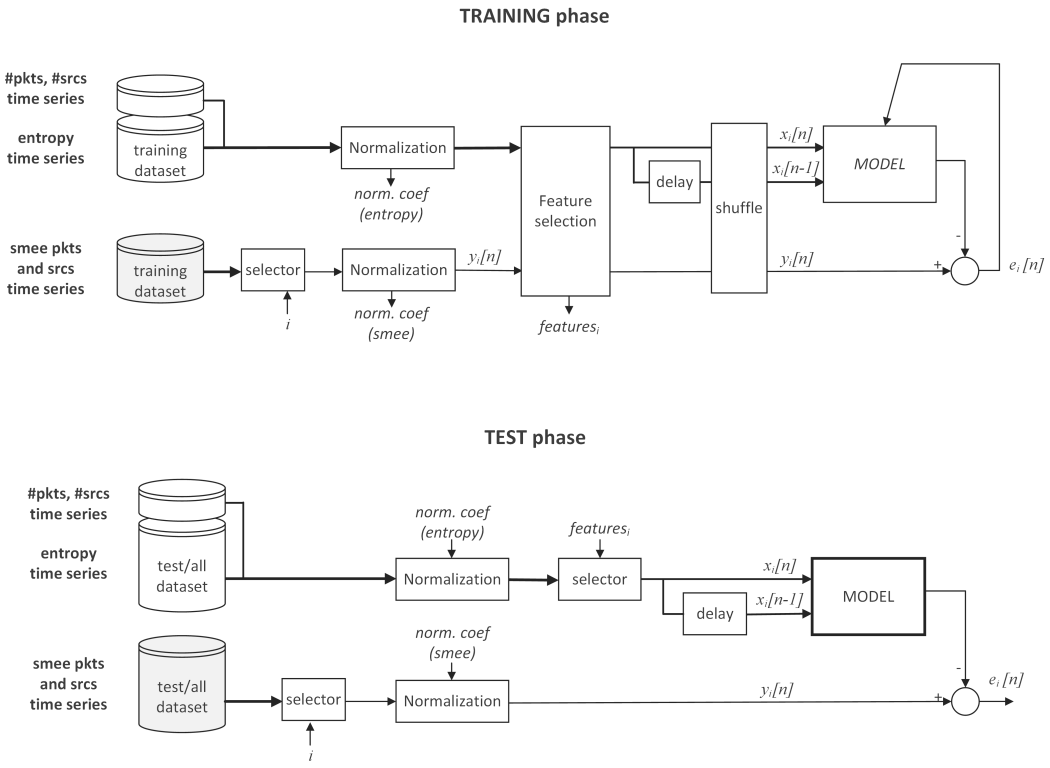


Figure 4.1: Describe in the caption exactly what can be seen in the figure

CHAPTER 5

Experiments

To inspect the potential benefits of self-supervised pre-training for ML-based intrusion detection we chose to take a look at LSTM and Transformer networks as they are suited to process sequences of variable length and have shown promising results in the past . Network traffic data can be looked at from a multitude of perspectives ranging from aggregate statistical data over different time-frames [MDES18] to looking at feature representations of single packets which can be viewed in the context of *flows*. Flows are loosely defined as sequences of packets that share a certain property [HBFZ19]. In our case we define flows as packets that share source and destination IP address, source and destination port, and the network protocol used. This creates the quintuple *<srcIP, dstIP, srcPort, dstPort, protocol>* as the key over which individual packets are aggregated to flows. We used the data pre-processing from [HBFZ19] as it fit the requirements for our experiments and was easily modifiable. The underlying data from which flow data is extracted are the *CIC-IDS-2017* [SLG18] and *UNSW-NB15* [MS15] NIDS datasets. After the data pre-processing from [HBFZ19] each packet is represented by source port, destination port, packet length, Interarrival Time (IAT), packet direction and all TCP flags (SYN, FIN, RST, PSH, ACK, URG, ECE, CWR, NS) resulting in 15 input features to be used in training the NNs.

give examples

The task of the NNs is to classify each flow into either *benign* or *attack* which results in a binary classification problem. Ordinary network traffic that should be ignored by the IDS is labeled as *benign* and flows that constitute or are part of a cyber-attack are labeled as *attack*. As there are only two possible labels, Binary Cross Entropy (BCE) can be used as loss function to determine the distance between the predicted label by the NNs and the actual label . For updating weights we use the *Adam* optimizer [KB14] which is an extension to the commonly used SGD method. Similar to *AdaGrad* [Rud16] and *RMSProp* [Rud16] it maintains separate learning rates for each individual weight instead of using the same learning rate for every weight like in classic SGD. Compared to other optimizers *Adam* was shown to be more effective in improving training efficiency

give more detailed explaination of BCE Loss

[KB14] and is appropriate for noisy or sparse gradients which can occur when working with RNNs in general.

As a premise for our research we trained the LSTM and the Transformer network in a solely supervised fashion to get a baseline later results can be compared to. Supervised training was performed for 100 epochs each for 90%, 5% and 1% of available data and a constant 10% of data for validation which has not been used for training. We specifically wanted to know how the networks would perform in a scenario where very little labeled training data was available as this would best describe a scenario where large amounts of unlabeled data are available for self-supervised pre-training and only a small amount of labeled data for fine tuning. To pre-train a NN the network is given a task that is not necessarily connected to the final purpose of the network, often referred to as a *proxy task*. By solving the proxy task the network attempts to find structure in the data and should learn to form a more abstract representation of the data within its latent space. E.g. with BERT pre-training is performed by masking a certain percentage of input tokens and having the NN predict the missing words and additionally letting the network guess whether one sentences precedes another in a text. We defined our own proxy tasks for pre-training the networks as described in the following sections. Pre-training is performed with 89% of available data, supervised fine-tuning with 1% and validation with 10% of data.

5.1 Self-supervised Pre-training for Long Short-Term Memory Networks

For our LSTM network we chose a three layer LSTM with a *hidden size* and *cell size* of 512. While a larger network might be more effective, this configuration proved to be swiftly trainable while also producing results close to the optimum provide numbers . Since we are only interested in comparisons between different training methods applied to the same model, it is not necessary to increase model size to achieve optimal results as this would unnecessarily increase the training time needed until the model converges. For training the LSTM model, each flow is considered one sample and each packet is one token. The tokens are processed by the model in chronological order, meaning packets with an earlier timestamp will be processed first. The timestamp however is not part of the feature representation but is considered for data pre-processing to order packets within flows. For pre-training the LSTM we devised five different proxy tasks for the model to solve in a self-supervised fashion: Predicting the next packet in the flow, predicting masked features of randomly chosen packets and predicting randomly masked packets, the identity function and an AutoEncoder. The Mean Absolute Error (MAE) is used to determine the divergence between prediction and target data. Translating to PyTorch this means we used *L1Loss* with *mean* reduction as the loss function for pre-training. We tuned the hyper-parameters of training for both supervised and self-supervised training to an initial *learning rate* of 10^{-3} and a *batch size* of 128. Over the training process, the learning rate will be adjusted by Adam so the model is robust to changes on the initial

learning rate.

write

- different parameterization of LSTM
- two consecutive 3-layered LSTMs
- orthogonal initialization
- CrossEntropy Loss instead of BCE

5.1.1 Identity Function

The simplest form of a proxy-task for pre-training is having the model learn the identity function. In practice that means that input sequence $x^{(t)}$ and target sequence $y^{(t)}$ are the same $x^{(t)} = y^{(t)} = x^{(1)}, x^{(2)}, ..., x^{(n)}$ where n is the sequence length. The model learns to convey the information through the network at each time step.

5.1.2 Predict Packet

For this proxy task, the model has to predict the next packet in the flow. We started by predicting only the last packet in each flow but then moved to predicting all packets in a flow except the first. This means having a *sequence-to-sequence* model where the inputs are all tokens in one flow with length n except the last, because it has no successor: $x^{(t)} = (x^{(1)}, x^{(2)}, ..., x^{(n-1)})$. The target data are all tokens in the same flow except the first, because it has no predecessor: $y^{(t)} = (x^{(2)}, x^{(3)}, ..., x^{(n)})$. LSTMs process data in sequential order so at each time step, the model only has information of packets in the past and is to predict what the next packet in the flow will be. This results in two comparable tensors $y^{(t)}$ and the model output sequence $\hat{y}^{(t)} = (\hat{y}^{(1)}, \hat{y}^{(2)}, ..., \hat{y}^{(n-1)})$ of equal length $n-1$ between which a differentiable loss $C(y^{(t)}, \hat{y}^{(t)})$ can be calculated. This way, a lot of information is conveyed to the network when compared to only predicting the last packet in a flow. At first glance, this looks similar to the identity function in 5.1.1. The key difference is however, that the token which is to be predicted is not yet available as an input token to the model, meaning it has to derive the features by other means than conveying the requested input token to the output. The loss is calculated as the MAE (*L1Loss* with *mean* reduction) between the predicted logits and the target data sequences.

5.1.3 Mask Features

For this pre-training task, the model is to predict masked features of some packets in the sequence. We have tried multiple masking values but -1 produces the best results out of the values we tried . This proxy task in particular can be parameterized in different ways. E.g. the number of features and which features to mask, if always the same features are

give a comparison of values

masked or if the selection is random for each packet or for each flow, if every packet in the sequence has some masked features or if there is only a chance that a packet is selected for masking. Those are only some examples of how this task can be set up in different ways. To be completely exhaustive was not possible, so we compiled a selection of some of the variations as an overview of the parameter space. For pre-training the model the masked data is provided as input sequence and the unmasked data is the target. The loss is calculated as the MAE (*L1Loss* with *mean* reduction) between the predicted *logits* and the target data sequences.

enumerate all parameter combinations used

5.1.4 Mask Packets

Similar to the pre-training in BERT, all features of random packets in the sequence are masked with a value of -1 and the model is to predict the masked tokens. Again, MAE is used as the loss function. Unlike to BERT, we don't only look at the masked tokens when calculating the loss but compare every feature of every packet, also the non-masked ones, which adds an auto-encoding property to the pre-training. We found this to have more beneficial effect on the results than only looking at the masked packets. The most important parameter here is the ratio of how many packets per sequence are to be masked compared to its sequence length. To work with an absolute number of masked packets is not feasible as sequence length varies from 1 to a set max sequence length which in our case was 100. If an absolute number was used to determine how many packets should be masked some sequences would be completely masked out which would not be beneficial for training.

5.1.5 Auto-Encoder

5.2 Self-supervised Pre-training for Transformer Networks

Following the example of BERT we only used the encoder part of the transformer since the decoder does not provide any benefit for classification problems. We tuned the model parameters to be 10 Transformer layers, each layer consisting of a 3-headed Multi-Head Attention block and a feed-forward network with a forward expansion of 20 times the input size, i.e. the number of features per packet. Since we did not observe any over-fitting during training, we set the drop-out rate to zero (except for training with the Auto-Encoder 5.2.2). Like with the LSTM we devised a series of proxy tasks for pre-training the model in self-supervised fashion. Since the information flow is different in Transformers than it is in LSTMs, the pre-training task *Predict Packets* 5.1 we used for the LSTM is no longer feasible. While the LSTM at each stage has only access to all the tokens it processed up to this point, the Transformer has access to all input tokens at each stage of the execution which is one of the benefits of self-attention [VSP+17a]. Contrary to our expectations, supervised training on the Transformer takes longer than on the LSTM to achieve the observed optimal accuracy of 99,65%. In other words, when

training the LSTM and the Transformer network for the same amount of time, the LSTM produces better results. In the following sections we describe the pre-training methods we used for to pre-train the Transformer network.

write

- two consecutive TransformerEncoders, one for pre-training, one for fine-tuning
- Classification (CLS) Token
- Resetting last Layers 1,...,5 of Transformer after pre-training
- Use decoder also
- different dropout rates
- different number of attention heads

5.2.1 Mask Features

Analogous to the *Mask Features* proxy task for the LSTM, we used the same method for pre-training the Transformer.

5.2.2 Autoencoder

Autoencoder are an established concept when it comes to self-supervised learning . With this method input and target data are the same and the network is tasked with reconstructing the input data at the output. To prevent the network from simply "transporting" the input tokens through the network without having to learn anything, a form of regularization is introduced to force the network into learning an abstract representation of the data [BKG21]. In our case, we used the dropout rate to introduce artificial noise into the input data.

give some examples

5.2.3 Mask Packet

For this proxy task, random packets in the flow are masked with a value of -1 and the model is to predict only the masked packets. Since a packet in a flow can be seen as a word in a sentence, and the feature representation of a packet is similar to an embedded word vector, this pre-training task is analogous to the method used in BERT [DCLT18].

CHAPTER 6

Results

- maximum accuracy with 0-90-10 pre-sup-val training
- comparison between pretraining accuracy with different proxy tasks for 10-80-10 pre-sup-val training
- comparison between pretraining accuracy with different proxy tasks for 1-89-10 pre-sup-val training
- comparison between pretraining accuracy with different proxy tasks for subset 10_flows subset pre-sup-val training
- comparison of performance improvements for different amounts of supervised training
- comparison of performance improvements for different compositions of pretraining data
- comparison between multiple datasets
- comparison to orthogonal initialization

6.1 Long Short-Term Memory Network

6.2 Transformer Network

6.3 Explainability

- close look at differences in performance for different attack classes

- partial dependency plots
- neuron activation

CHAPTER 7

Discussion

Discuss any open issues and give a critical reflection of your work. E.g., what could be problems to deploy your method or do you have an idea how your findings could be generalized or what could be a hindrance for generalization?

Also discuss strange things you observed or results you could not completely explain.

CHAPTER 8

Conclusion

Conclude your work. Stress again what was the contribution. Provide an outlook what could be further improvements and what could future research do to continue your work.

APPENDIX A

Rules for writing the Thesis

A.1 General Rules

- Code of Conduct: You need to understand and sign the TU Code of Conduct before working on a thesis at TU. You can find it athttps://www.tuwien.at/fileadmin/Assets/dienstleister/Datenschutz_und_Dokumentenmanagement/Code_of_Conduct_fuer_wissenschaftliches_Arbeiten.pdf
- Time Planning: Plan your thesis realistically. Check how much time you need for studies and work and other obligations to estimate how much time you can spend per week on your thesis. Especially if you have to learn new things (theoretical knowledge in a new field, a new tool, a new programming language), plan sufficient time for this. Keep in mind that always unforeseen problems can occur. So plan some buffer time.
- External Deadlines: Make all deadlines clear before you start the thesis. E.g. if you have any time constraints wrt. projects, visa applications, planned employment or any other time restrictions in your studies, let the supervisor know this before you start working on the thesis. Last minute request will not be accepted.
-
-

A.2 Writing the Thesis

- Use the CN group latex Master Thesis template
- Continuously document what you are doing

  - Make notes about papers you read
  - Document all experiment details.Also if experiments are not successful it is important to document what you did and which errors occurred
  - Document your software in a way that others can continue to understand and modify/extend the software

- Use US english
- Consider to write a paper from your results

A.2.1 Tenses

- Use present tense for state of art
-

TZ TODO: add rules and references about tenses

A.2.2 References, Copyright and Citations

- citations need to be clearly marked (see code of conduct)
- no re-phrasing
- Ideally use no figures copied from somewhere else. If figure are copied, a) the copyright must allow use it and b) they have to be correctly cited
- You may use sherpa to identify the copyright rules for particular Journal. `http://www.sherpa.ac.uk/romeo/index.php?la=en&fIDnum=|&mode=simple`
- Some useful definitions and rules for plagiarism and self-plagiaraism can be found at `https://www.fsdr.at/plagiarism`
- Rules how to correctly cite a creative commons figure see `https://commons.wikimedia.org/wiki/Commons:Reusing_content_outside_Wikimedia`
- References: Use books or scientific papers as reference instead of web pages or blog entries
- If you have to cite a web page you have to provide the date when you last accessed the page , last accessed at YYYY-MM-DD

TZ TODO: add example for creative commons reference

TZ TODO: add references to code of conduct and plagiarism rules

A.2.3 Latex Tools

TZ TODO: Add links

A.3 Tools and Infrastructure

The following tools are useful:

- thesis template
- zotero (`zotero.org`): Tool for collecting papers and sharing papers with others (creating a zotero group)
- SVN or git for joint paper editing
- Overleaf for short term joint editing of latex files

Open Issues

- Getting data sets from CN group
- Getting access to CN infrastructure (compute cluster, GPU, storage)
- Access to NTARC?
- provide a template for describing experiments

A.4 Communication

The first rule is to stay in contact and inform the supervisor(s) about your progress, questions and difficulties.

So always ask:

- If anything is not clear about what you should do
- If you do not understand something (e.g., a paper, an equation, a statement)
- If you have problems with software, programming, etc.
- If you don't know which papers are relevant and which not
- If you have a new idea or want to take a different path.

Further rules:

- Friday updates: send a brief update to your supervisor(s) every Friday. You can include any ideas, questions or difficulties that you had during the week. If you did not make any progress in the week just send an email saying that you did not make progress.
- Use an SVN or git repository to store the latest version of your document
- Use meaningful file names: Example: YYYY-MM-DD-YourLastName-DocumentName-version
- Send an email to supervisors(s) if a new version to be reviewed is in the SVN
- clearly mark all changes in the document that you made compared to the last version. Show how you addressed comments.

A.5 Reproducibility

A.6 Publishing Papers

A.6.1 Finding suitable Conferences

Top Conferences and Journals

Conferences and Journal Rankings

A.6.2 Using arxiv

A.7 Open Issues

- put change marking method in template

APPENDIX B

Introduction to LaTeX

Since LaTeX is widely used in academia and industry, there exists a plethora of freely accessible introductions to the language. Reading through the guide at `https://en.wikibooks.org/wiki/LaTeX` serves as a comprehensive overview for most of the functionality and is highly recommended before starting with a thesis in LaTeX.

B.1 Installation

A full LaTeX distribution consists not only of the binaries that convert the source files to the typeset documents, but also of a wide range of packages and their documentation. Depending on the operating system, different implementations are available as shown in Table B.1. **Due to the large amount of packages that are in everyday use and due to their high interdependence, it is paramount to keep the installed distribution up to date.** Otherwise, obscure errors and tedious debugging ensue.

B.2 Editors

A multitude of TeX editors are available differing in their editing models, their supported operating systems and their feature sets. A comprehensive overview of editors can be

Distribution	Unix	Windows	MacOS
TeX Live	**yes**	yes	(yes)
MacTeX	no	no	**yes**
MikTeX	(yes)	**yes**	yes

Table B.1: TeX/LaTeX distributions for different operating systems. Recomended choice in **bold**.

	Description
1	Scan for refs, toc/lof/lot/loa items and cites
2	Build the bibliography
3	Link refs and build the toc/lof/lot/loa
4	Link the bibliography
5	Build the glossary
6	Build the acronyms
7	Build the index
8	Link the glossary, acronyms, and the index
9	Link the bookmarks

	Command
1	`pdflatex.exe  example`
2	`bibtex.exe  example`
3	`pdflatex.exe  example`
4	`pdflatex.exe  example`
5	`makeindex.exe -t example.glg -s example.ist -o example.gls example.glo`
6	`makeindex.exe -t example.alg -s example.ist -o example.acr example.acn`
7	`makeindex.exe -t example.ilg -o example.ind example.idx`
8	`pdflatex.exe  example`
9	`pdflatex.exe  example`

Table B.2: Compilation steps for this document. The following abbreviations were used: table of contents (toc), list of figures (lof), list of tables (lot), list of algorithms (loa).

found at the Wikipedia page `https://en.wikipedia.org/wiki/Comparison_of_TeX_editors`. TeXstudio (`http://texstudio.sourceforge.net/`) is recommended. Most editors support a synchronization of the generated document and the LaTeX source by `Ctrl` clicking either on the source document or the generated document.

B.3 Compilation

Modern editors usually provide the compilation programs to generate Portable Document Format (PDF) documents and for most LaTeX source files, this is sufficient. More advanced LaTeX functionality, such as glossaries and bibliographies, needs additional compilation steps, however. It is also possible that errors in the compilation process invalidate intermediate files and force subsequent compilation runs to fail. It is advisable to delete intermediate files (`.aux`, `.bbl`, etc.), if errors occur and persist. All files that are not generated by the user are automatically regenerated. To compile the current document, the steps as shown in Table B.2 have to be taken.

B.4 Basic Functionality

In this section, various examples are given of the fundamental building blocks used in a thesis. Many LaTeX commands have a rich set of options that can be supplied as optional arguments. The documentation of each command should be consulted to get an impression of the full spectrum of its functionality.

B.4.1 Floats

Two main categories of page elements can be differentiated in the usual LaTeX workflow: *(i)* the main stream of text and *(ii)* floating containers that are positioned at convenient positions throughout the document. In most cases, tables, plots, and images are put into such containers since they are usually positioned at the top or bottom of pages. These are realized by the two environments `figure` and `table`, which also provide functionality for cross-referencing (see Table B.3 and Figure B.1) and the generation of corresponding entries in the list of figures and the list of tables. Note that these environments solely act as containers and can be assigned arbitrary content.

B.4.2 Tables

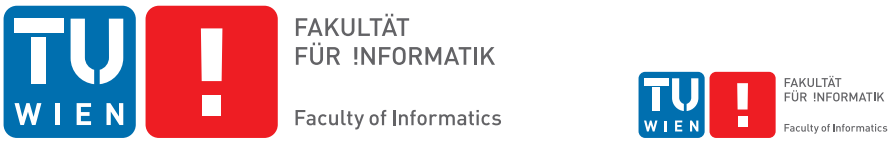
A table in LaTeX is created by using a `tabular` environment or any of its extensions, e.g., `tabularx`. The commands `\multirow` and `\multicolumn` allow table elements to span multiple rows and columns.

Position		
Group	Abbrev	Name
Goalkeeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duburry
	DC	Dominic Matteo
	RB	Didier Domi
Midfielders	MC	David Batty
	MC	Eirik Bakke
	MC	Jody Morris
Forward	FW	Jamie McMaster
Strikers	ST	Alan Smith
	ST	Mark Viduka

Table B.3: Adapted example from the LaTeXguide at `https://en.wikibooks.org/wiki/LaTeX/Tables`. This example uses rules specific to the `booktabs` package and employs the multi-row functionality of the `multirow` package.

B.4.3 Images

An image is added to a document via the `\includegraphics` command as shown in Figure B.1. The `\subcaption` command can be used to reference subfigures, such as Figure B.1a and B.1b.



(a) The header logo at text width. (b) The header logo at half the text width.

Figure B.1: The header logo at different sizes.

B.4.4 Mathematical Expressions

One of the original motivation to create the TeX system was the need for mathematical typesetting. To this day, LaTeX is the preferred system to write math-heavy documents and a wide variety of functions aids the author in this task. A mathematical expression can be inserted inline as $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ outside of the text stream as

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

or as numbered equation with

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}. \tag{B.1}$$

B.4.5 Pseudo Code

The presentation of algorithms can be achieved with various packages; the most popular are `algorithmic`, `algorithm2e`, `algorithmicx`, or `algpseudocode`. An overview is given at `https://tex.stackexchange.com/questions/229355`. An example of the use of the `alogrithm2e` package is given with Algorithm B.1.

B.5 Bibliography

The referencing of prior work is a fundamental requirement of academic writing and well supported by LaTeX. The BibTeX reference management software is the most commonly used system for this purpose. Using the `\cite` command, it is possible to reference entries in a `.bib` file out of the text stream, e.g., as [Tur36]. The generation of the formatted bibliography needs a separate execution of `bibtex.exe` (see Table B.2).

Algorithm B.1: Gauss-Seidel

Input: A scalar ϵ, a matrix $\mathbf{A} = (a_{ij})$, a vector $\vec{b}$, and an initial vector $\vec{x}^{(0)}$
Output: $\vec{x}^{(n)}$ with $\mathbf{A}\vec{x}^{(n)} \approx \vec{b}$

```
for k ← 1 to maximum iterations do
    for i ← 1 to n do
        x_i^(k) = (1/a_ii) (b_i − Σ_{j<i} a_ij x_j^(k) − Σ_{j>i} a_ij x_j^(k−1));
    end
    if |x^(k) − x^(k−1)| < ε then
        break for;
    end
end
return x^(k);
```

B.6 Table of Contents

The table of contents is automatically built by successive runs of the compilation, e.g., of `pdflatex.exe`. The command `\setsecnumdepth` allows the specification of the depth of the table of contents and additional entries can be added to the table of contents using `\addcontentsline`. The starred versions of the sectioning commands, i.e., `\chapter*`, `\section*`, etc., remove the corresponding entry from the table of contents.

B.7 Acronyms / Glossary / Index

The list of acronyms, the glossary, and the index need to be built with a separate execution of `makeindex` (see Table B.2). Acronyms have to be specified with `\newacronym` while glossary entries use `\newglossaryentry`. Both are then used in the document content with one of the variants of `\gls`, such as `\Gls`, `\glspl`, or `\Glspl`. Index items are simply generated by placing `\index{`⟨*entry*⟩`}` next to all the words that correspond to the index entry ⟨*entry*⟩. Note that many enhancements exist for these functionalities and the documentation of the `makeindex` and the `glossaries` packages should be consulted.

B.8 Tips

Since TEX and its successors do not employ a What You See Is What You Get (WYSIWYG) editing scheme, several guidelines improve the readability of the source content:

- Each sentence in the source text should start with a new line. This helps not only the user navigation through the text, but also enables revision control systems

(e.g. Subversion (SVN), Git) to show the exact changes authored by different users. Paragraphs are separated by one (or more) empty lines.

- Environments, which are defined by a matching pair of `\begin{name}` and `\end{name}`, can be indented by whitespace to show their hierarchical structure.
- In most cases, the explicit use of whitespace (e.g. by adding `\hspace{4em}` or `\vspace{1.5cm}`) violates typographic guidelines and rules. Explicit formatting should only be employed as a last resort and, most likely, better ways to achieve the desired layout can be found by a quick web search.
- The use of bold or italic text is generally not supported by typographic considerations and the semantically meaningful `\emph{...}` should be used.

The predominant application of the LaTeX system is the generation of PDF files via the PdfLaTeX binaries. In the current version of PdfLaTeX, it is possible that absolute file paths and user account names are embedded in the final PDF document. While this poses only a minor security issue for all documents, it is highly problematic for double blind reviews. The process shown in Table B.4 can be employed to strip all private information from the final PDF document.

	Command
1	Rename the PDF document `final.pdf` to `final.ps`.

2	Execute the following command:
	`ps2pdf -dPDFSETTINGS#/prepress ^`
	`-dCompatibilityLevel#1.4 ^`
	`-dAutoFilterColorImages#false ^`
	`-dAutoFilterGrayImages#false ^`
	`-dColorImageFilter#/FlateEncode ^`
	`-dGrayImageFilter#/FlateEncode ^`
	`-dMonoImageFilter#/FlateEncode ^`
	`-dDownsampleColorImages#false ^`
	`-dDownsampleGrayImages#false ^`
	`final.ps final.pdf`

On Unix-based systems, replace # with = and ^ with \.

Table B.4: Anonymization of PDF documents.

B.9 Resources

B.9.1 Useful Links

In the following, a listing of useful web resources is given.

https://en.wikibooks.org/wiki/LaTeX An extensive wiki-based guide to LaTeX.

http://www.tex.ac.uk/faq A (huge) set of Frequently Asked Questions (FAQ) about TeX and LaTeX.

https://tex.stackexchange.com/ The definitive user forum for non-trivial LaTeX-related questions and answers.

B.9.2 Comprehensive TeX Archive Network (CTAN)

The CTAN is the official repository for all TeX related material. It can be accessed via `https://www.ctan.org/` and hosts (among other things) a huge variety of packages that provide extended functionality for TeX and its successors. Note that most packages contain PDF documentation that can be directly accessed via CTAN.

In the following, a short, non-exhaustive list of relevant CTAN-hosted packages is given together with their relative path.

algorithm2e Functionality for writing pseudo code.

amsmath Enhanced functionality for typesetting mathematical expressions.

amssymb Provides a multitude of mathematical symbols.

booktabs Improved typesetting of tables.

enumitem Control over the layout of lists (`itemize`, `enumerate`, `description`).

fontenc Determines font encoding of the output.

glossaries Create glossaries and list of acronyms.

graphicx Insert images into the document.

inputenc Determines encoding of the input.

l2tabu A description of bad practices when using LaTeX.

mathtools Further extension of mathematical typesetting.

memoir The document class on upon which the `vutinfth` document class is based.

multirow Allows table elements to span several rows.

pgfplots Function plot drawings.

pgf/TikZ Creating graphics inside LaTeX documents.

subcaption Allows the use of subfigures and enables their referencing.

symbols/comprehensive A listing of around 5000 symbols that can be used with LaTeX.

voss-mathmode A comprehensive overview of typesetting mathematics in LaTeX.

xcolor Allows the definition and use of colors.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BKG21] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[HBFZ19] Alexander Hartl, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for rnns. *CoRR*, abs/1912.09855, 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[Lip15] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[MDES18] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. *CoRR*, abs/1802.09089, 2018.

[MS15] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). 11 2015.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SLG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 108–116. INSTICC, SciTePress, 2018.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:345–363, 1936.

[VSP+17a] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[VSP+17b] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[YSHZ19] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019.