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 Long Short-Term Memory (LSTM) and the Transformer networks as they are suited to process data 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 a feature representation of single packets which can be looked at 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] Network Intrusion Detection System (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 into 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 Stochastic Gradient Descent (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

give more detailed explaination of BCE Loss in improving training efficiency [KB14] and is appropriate for noisy or sparse gradients which can occur when working with Recurrent Neural Networks (RNN) 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 the pre-training results can be compared to. Supervised training was performed for 10 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 the input 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.

## 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 . 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 the packets within the flow. For pre-training the LSTM we devised three different proxy tasks for the model to solve in a self-supervised fashion: Predicting the next packet in the flow, predicting masked features where the same feature is masked in every packet of the sample and predicting randomly masked packets. The Mean Absolute Error (MAE) is used to determine the error 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.

provide numbers

### 5.1.1 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 except the last, because it has no successor. The target data are all tokens in the same flow except the first, because it has no predecessor. This results in two comparable tensors of equal length $n - 1$ where $n$ is the original sequence length of the flow. This way, a lot more information is conveyed to the network when compared to only predicting the last packet in a flow. At first glance, this looks similar to Auto-Encoding. 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.

### 5.1.2 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 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.

give a comparison of values

enumerate all parameter combinations

### 5.1.3 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.

## 5.2 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+17]. 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.

### 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

give some examples

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.

### 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 method is analogous to the method to pre-train the BERT model.