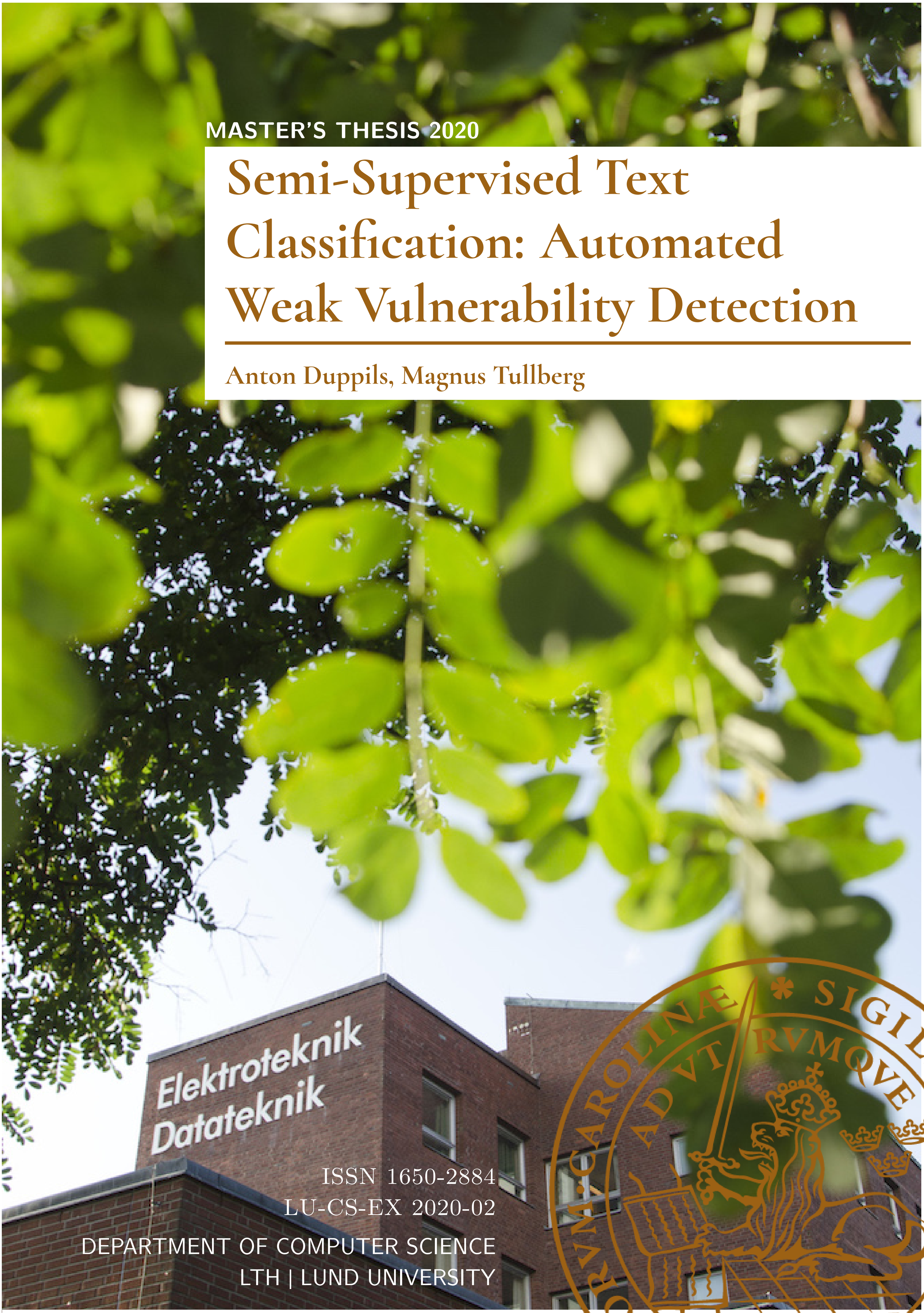
MASTER'S THESIS 2020

# Semi-Supervised Text Classification: Automated Weak Vulnerability Detection

Anton Duppils, Magnus Tullberg



ISSN 1650-2884
LU-CS-EX 2020-02
DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-02

# Semi-Supervised Text Classification: Automated Weak Vulnerability Detection

**Anton Duppils, Magnus Tullberg**

# Semi-Supervised Text Classification: Automated Weak Vulnerability Detection

Anton Duppils
dat13adu@cs.lth.se

Magnus Tullberg
mat12mtu@cs.lth.se

January 28, 2020

Master's thesis work carried out at Debricked AB.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Emil Wåreus, emil.wareus@debricked.com

Examiner: Pierre Nugues, pierre.nugues@cs.lth.se

## Abstract

Open-source software is prevalent in the development of new technologies. Monitoring software updates for vulnerabilities is expensive and time-consuming. Online discussions surrounding new software updates can often provide vital information regarding emerging risks. In this Master's thesis, we present a novel approach for automating surveillance of software through the use of natural language processing methods on open-source issues. We explore the potential of virtual adversarial training, a popular semi-supervised learning technique to leverage the vast amounts of unlabeled data available to achieve improved performance. On industry data, the best performing model is a hierarchical attention network with virtual adversarial training that utilizes the innate document structure to encapsulate the text. Promising results are achieved for text classification in the computer security domain.

**Keywords**: NLP, SSL, HAN, VAT, Security, Vulnerability Detection, Text Classification

# Acknowledgements

This Master's thesis would not be possible if it were not for the valuable advice from our supervisors Emil Wåreus and Marcus Klang. They both went beyond the call of duty with a great deal of support and engagement. Thanks for taking the time to discuss our ambitious ideas and for keeping us grounded. Having two supervisors meant that we were blessed with different opinions, pushing us to consider far more options than anticipated. We also want to thank Debricked for supplying us with peripherals, a nice work environment, and sponsoring for model training. We would also like to take this moment to thank our friend Alex for keeping us in good spirits and for the occasional good tip. Lastly, we appreciate the feedback and critique from our examiner Pierre Nugues and hope the final revision of this Master's thesis does the topic justice.

# Contents

# Chapter 1

# Introduction

## 1.1 Task

The use of open-source software has proliferated in modern times. According to an Open Source Security and Risk Analysis report by Synopsys, 96% of codebases scanned in 2018 used open-source code (Synopsys, 2018). A followup report in 2019 shows an increase in open-source usage to more than 99%. Vulnerabilities in open-source components are often mismanaged as the same report also highlights that 40% of the aforementioned codebases feature open-source vulnerabilities that are more than 10 years old (Synopsys, 2019).

Open-source updates can expose security vulnerabilities. Keeping track of vulnerabilities in open-source software can help mitigate the potential damage done by malicious parties. It is hard to keep track of when a new vulnerability has been discovered. Human resources dedicated to vulnerability tracking is expensive and has limited reach. It has been found that 90% of exploited exposures are from previously known issues (Ferenc et al., 2019). Therefore it is decidedly useful to be able to detect reported vulnerabilities in text. An example of a truncated computer security related sentence can be viewed in Table 1.1. Automated weak vulnerability detection using text classification on discussions in open-source repositories could potentially provide awareness of security flaws. This thesis explores the potential for automation with the goal of providing non-trivial classification of computer security discussion.

## 1.2 Contribution

The work in this Master's thesis explores the possibilities of text classification in the domain of computer security. The domain is still in its early stages and the best model architectures and general approaches have not been established. This study attempts to construct benchmarks in the domain for future works to compare against. The results prove that the problem

| id2 | **Examples of data in training set** (truncated) |
|---|---|
| **Security related** | |
| CVE-2018-11392 | An arbitrary file upload vulnerability in classes/... |
| **Non-security related** | |
| docker docker.github.io 8022 | Add screenshot for backup warning, minor edits... |

**Table 1.1:** Table containing a truncated example of a security related example and a non-security related example.

is indeed solvable with natural language processing (NLP) and achieve quite respectable performance on binary text classification. The HAN model architecture, first proposed by Yang et al. (2016), attempts to make use of the innate structure of text and is the primary model proposed for this task. The semi-supervised learning technique Virtual adversarial training (VAT) (Miyato et al., 2016) is used to leverage the large quantities of unlabeled data acquired. The use of machine learning in the computer security domain is intended to alleviate the great cost of human resources in monitoring open-source projects for potential vulnerabilities. Automation improves the coverage for vulnerability management. A quicker response is also possible, limiting damage. The best achieved performance for prediction on vulnerabilities is 97% precision with 49% recall on the main test set, achieving an F1 score of 65%. The best overall performance across several datasets is our HAVAN model, combining HAN with VAT.

The co-authors both contributed equally to this project. They have both been part of every step from start to finish.

## 1.3 Outline

The Master's thesis is divided into sections, in order: Theory, Method, Results, Discussion, and Conclusion. Theory handles the theoretical groundwork which the thesis builds its approach on and discusses previous work that inspired this research. A well educated NLP data scientist should be able to skip this section. The following section, Method, describes the workflow and process from the start of the thesis to its completion. The Results section presents the evaluation plots and tables. The predictions are made on several test datasets using both a baseline model from a recently published previous work with a convolutional neural network (CNN) model and our own HAN implementation with and without Virtual Adversarial Training (VAT). The results are elaborated upon in the Discussion section. The methodology, approaches used, and the potential sources of errors are discussed in detail. In the Conclusion section, the thesis reflects on how it has contributed to research, how these results can affect the industry, and what future work could improve the results and further advance the field.

# Chapter 2
# Related Work

## 2.1 Security Identification

Zou et al. (2018) present a model they call Security Bug Report Identifier (SBRer). The model is trained on labeled datasets and is specifically trained to distinguish between security related bug reports and non-security related bug reports. SBRer uses both textual features and meta features to try to maximize the identification rate. The SBRer is trained on a dataset consisting of 23,608 bug reports from Bugzilla using three different open-source products; Firefox, Seamonkey, and Thunderbird. The results achieved by the SBRer was with the precision of 99.4% and a recall of 79.9%.

Behl et al. (2014) propose a model that uses text mining approaches in combination with TF-IDF. The model tries to predict the nature of a bug to decide whether it is a security bug report or not using naïve bayes.

Though there is various research and related work on identifying bug reports from non-related bug reports, the research found on detecting if a text concerns security-related issues were sparse.

A new study exploring the potentials of natural language processing for security topic classification was published by Palacio et al. (2019), the creators of the Alpha SecureReqNet (SRN) model. The paper claims that the task of identifying security related texts is achievable but lacks benchmarks or comparisons with any previous works. The authors left a more extensive evaluation with several baseline models to be done in the future.

We took advantage of the opportunity to use their model as a benchmark neural network to compare our HAN model to. An open-source variant of the SRN model architecture is available for free online and contains most of the necessary code. SRN is a CNN as opposed to the more common recurrent neural networks (RNNs) used for problems in the text domain. CNNs have widespread use in image tasks, but did not have the same levels of success in text tasks until recently. The theoretical background for CNNs can be found in the appropriate Theory section 5.2.1 as well as how text problems are structured and fed into CNN architectures.

## 2.2 Document Classification

HAN (Yang et al., 2016) attempts to take advantage of sentence based structure in text. It is built using attention mechanisms and RNNs. HAN is developed specifically to work well for document classification.

## 2.3 Semi-Supervised Learning

There are several interesting SSL techniques. Most of these methods have been initially developed for image-based tasks in mind and some of them have been adjusted to work well with text-based problems. The purpose of SSL is to leverage the vast amount of unlabeled data that is often available for training better machine learning models.

Adversarial methods are a popular way to improve a model by creating training data that aims to trick the classifier into making wrong predictions.

### 2.3.1 Adversarial Networks

Generative adversarial networks have a generator and a judge. The generator creates fake images to feed to the judge. Both generated images and real images are fed to the judge and the judge tries to predict what images are real(Goodfellow et al., 2014a). This scheme improves both the generator and the judge in tandem by pitting them against each other. An alternative method that has found success on text problems is the discriminative adversarial network(dos Santos et al., 2017). The network has a predictor and a judge and the predictor labels unlabeled data and sends the annotated data to the judge. The judge must decide if the annotation was done by a human or by the predictor, leading to a similar adversarial problem that improves both predictor and judge.

### 2.3.2 Virtual Adversarial Training

Virtual adversarial training (VAT) (Miyato et al., 2016) is another method first developed with image tasks in mind that has found relevance in text problems. VAT on text perturbs word embeddings in a direction that will have the highest chance of tricking the classifier into making the wrong prediction.

### 2.3.3 Self Learning

Self-learning, also called pseudo labeling, is a method of having the classifier make predictions for an unlabeled dataset and then adding it into the pool of labeled training data with the classifier's annotation. This type of method incurs a certain risk of overfitting to a certain subset of data, but has had some recent success from Xuan et al. (2017) where it was used with a naive Bayes classifier for assigning the correct developers to each bug report.

### 2.3.4 Variational Autoencoders

Variational autoencoders have been used recently on the SSL text classification problem by Xu et al. (2016) with a promising degree of success. The model consists of an encoder and a decoder. The encoder maps the input text to a latent space of lower dimension and the decoder is responsible for mapping values in this space back to human language. Encoding and decoding data can lead to loss, a reconstruction error, meaning that the input data will not be equal to the output data. In autoencoders, the encoder and decoder are made of neural networks aiming to learn the optimal encoding and decoding behavior by minimizing the reconstruction error. Variational Autoencoders build on the concept of autoencoders by regularizing the latent space so the decoder can be used on a random point in latent space to generate data of acceptable quality(Rocca, 2019).

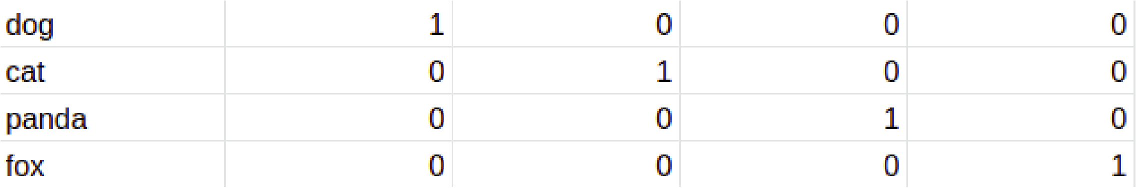
# Chapter 3
# Theory

## 3.1 Language Model

Language modeling is a way of learning the innate structure of a language. Since language has a restrictive rule-set, the language model data is sparse. Most combinations of words do not form an acceptable sentence. There are many ways of building a language model for word representations. In this study, we have tried 100 dimensional GloVe and security domain SecureReqNnet embeddings.

### 3.1.1 Word Representation

A simple word representation scheme is *one-hot encoding*. It constructs a matrix with dimensions corresponding to the number of unique words and the number of input data. Each row contains the number one for each unique word that occurred in the input and zero for all other words. Since most words will not appear in any given text input, the matrix is sparse. This carries with it the curse of dimensionality, which are problems that scale poorly with high-dimensional representations.



| | | | | |
|---|---|---|---|---|
| dog | 1 | 0 | 0 | 0 |
| cat | 0 | 1 | 0 | 0 |
| panda | 0 | 0 | 1 | 0 |
| fox | 0 | 0 | 0 | 1 |

**Figure 3.1:** Example of one-hot encoded vectors. The words are represented with 0 at each index except one with the value 1. The total words in the one-hot encoding is equal to the dimension or length of the vector.

### 3.1.2 Word Embedding

Word embedding is defined as language modeling and feature learning techniques in NLP that map symbols (words) into a vector space. This vector space has some desirable properties, such as similarity by angle and allowing dense representation. Dense representations generally have less computational cost than one-hot encoding when working with large inputs and vocabularies. Since the dimensions are fixed, it does not suffer from the curse of dimensionality. Embeddings can represent the similarity or distinctness of words, often proving helpful in NLP tasks. Note the classic example:

The words "king", "man", and "woman" are selected. If we take the embedding values of "king" and subtract the embedding for "man" and add "woman" the result will be the embedding for "queen". We note that one aspect being measured is the royal attribute, the other is gender. Word embedding can learn to represent these attributes so that words with similar attributes are close in space of a given dimension. See Figure 3.2 for a visual representation. This scenario assumes that one of the embedding dimensions has learned the attribute gender and one has learned the attribute royal.

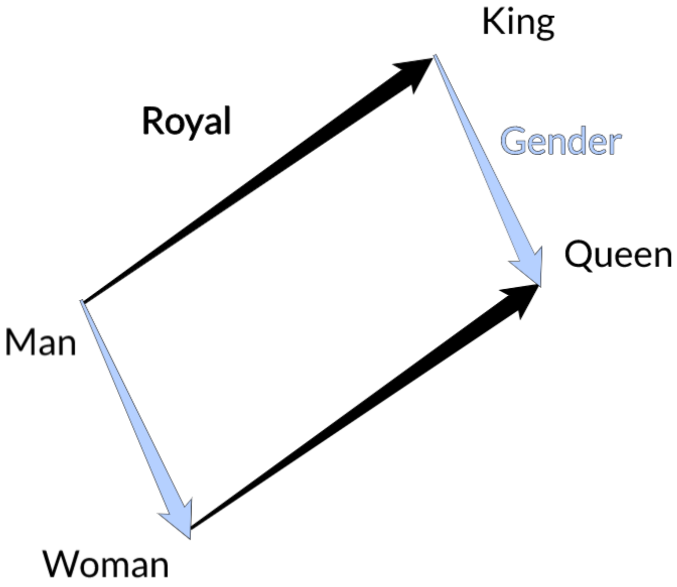


**Figure 3.2:** Embedding representation of words in 2d space where the dimensions were arbitrarily chosen to show the relations for royalty and gender. As can be seen from the figure above, an equal change in embedding values for man and woman lead to logical representations for king and queen respectively. Similarly, a change in the gender in terms of embedding perturbation can take the word man or king to woman or queen respectively.

The choice of dimensions for word embeddings is not necessarily intuitive. One may think that just increasing the dimensions of embeddings lead to better results, but more dimensions means a larger feature space, introducing the curse of dimensionality. Many common pretrained embeddings available typically have about 50 to 300 dimensions(J. Pennington, 2014).

It is common practice to randomize embedding initialization of words that are not in the vocabulary from a distribution with a certain mean and standard deviation. Randomly initialized embeddings are not much worse than pretrained embeddings for neural networks since the network will often learn the relations after some time regardless(Kocmi and Bojar, 2017).

Two common methods used to train word embeddings are continuous bag of words (CBOW) and skipgram. CBOW uses the frequency of the surrounding words to predict a word, which means CBOW predicts a missing word from a given context. Skipgram, on the other hand, uses a given word to predict the surrounding words, meaning skipgram predicts the context given a word. See figure 3.3 and 3.4 for an example.

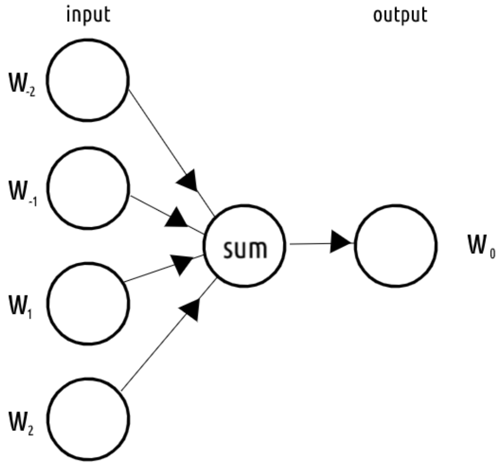


**Figure 3.3:** Bag of words as can be seen in the illustration, takes n words as input and calculates a prediction for which word is in the middle.

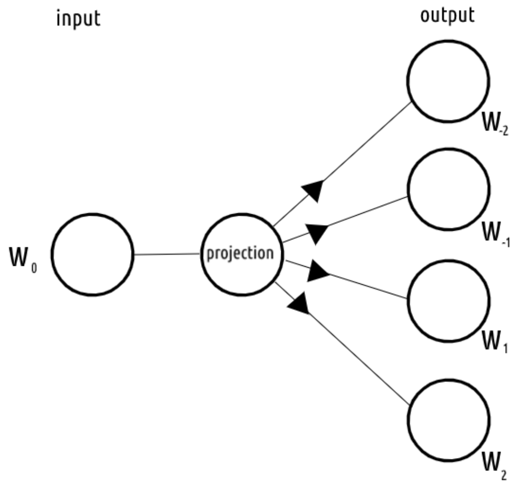


**Figure 3.4:** Skipgram takes one word and tries to predict the n surrounding words.

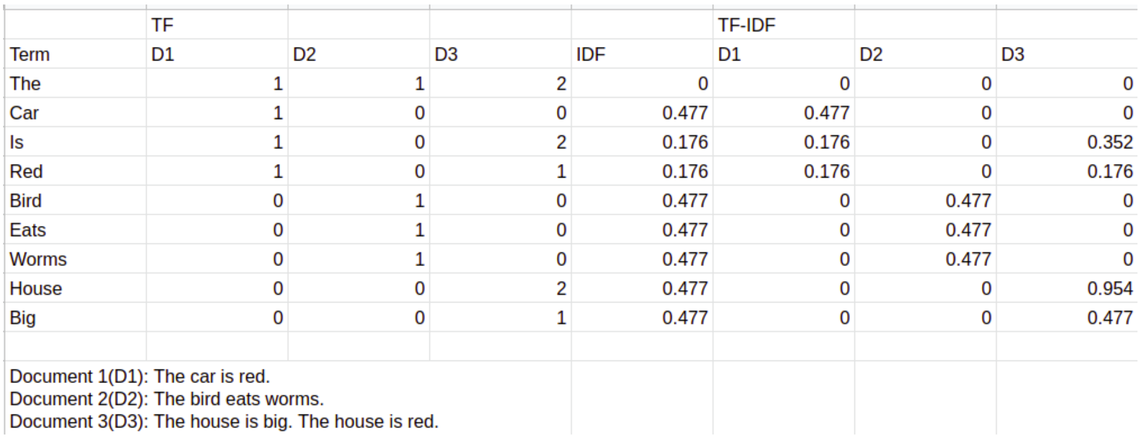
### 3.1.3 Term Frequency-Inverse Document Frequency

Term frequency-inverse document frequency (tf-idf) is a calculation on how important a term $t$ is to a document $d$ in a corpus $D$. The basics of it is built upon two bases, term frequency (TF) and inverse document frequency (IDF). TF is the count of a term $t$ in a document $d$. For a document $d$ containing the term $t$ $i$ times, the basic approach to TF would be to use the number of occurrences $i$. Often an approach that takes into account the length of the document may be used, such as dividing the basic TF by the number of words in the document thus normalizing it for each document. To compensate that the TF emphasizes more on common words, the IDF instead measures how much information the term provides by looking at the whole corpus. The IDF therefore emphasizes on the more interesting terms of the corpora, the terms which are more unique. The formula for IDF is

$$\log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where |D| is the number of documents and $\{d \in D : t \in d\}|$ is the number of documents the term $t$ appears in.

TF-IDF is the product of TF and IDF. An example of TF-IDF can be seen in Figure 3.5.



| | TF | | | | TF-IDF | | |
|---|---|---|---|---|---|---|---|
| Term | D1 | D2 | D3 | IDF | D1 | D2 | D3 |
| The | 1 | 1 | 2 | 0 | 0 | 0 | 0 |
| Car | 1 | 0 | 0 | 0.477 | 0.477 | 0 | 0 |
| Is | 1 | 0 | 2 | 0.176 | 0.176 | 0 | 0.352 |
| Red | 1 | 0 | 1 | 0.176 | 0.176 | 0 | 0.176 |
| Bird | 0 | 1 | 0 | 0.477 | 0 | 0.477 | 0 |
| Eats | 0 | 1 | 0 | 0.477 | 0 | 0.477 | 0 |
| Worms | 0 | 1 | 0 | 0.477 | 0 | 0.477 | 0 |
| House | 0 | 0 | 2 | 0.477 | 0 | 0 | 0.954 |
| Big | 0 | 0 | 1 | 0.477 | 0 | 0 | 0.477 |

Document 1(D1): The car is red.
Document 2(D2): The bird eats worms.
Document 3(D3): The house is big. The house is red.

**Figure 3.5:** TF-IDF example with a simple term frequency (TF), inverse document frequency (IDF), and term frequency - inverse document frequency (TF-IDF). Inverse document frequency is calculated using all documents and represents how rare the term is in the context of how many different documents contains the term.

## 3.2 Dimensionality Reduction

Dimensionality Reduction serves to find a representation for certain data that retains as much of the important information as possible, while reducing the number of dimensions. A more succinct representation allows for faster calculations. It can also improve human understanding of data through plotting the observations in 2 or 3 dimensions. In this section, we present a variety of methods and the theory for which these methods are based on. The methods used in this thesis are: Latent Semantic Analysis, T-Distributed Stochastic Neighbor Embedding, and Uniform Manifold Approximation and Projection.

### 3.2.1 Truncated Singular Value Decomposition

When working with highly sparse matrices, it is often desirable to reduce the dimensionality of the matrix, making it dense. One common way is to use Truncated Singular Value Decomposition (TruncSVD), to do both.

TruncSVD is an approximation of the Singular Value Decomposition (SVD) of a matrix, containing only the $k$ largest singular values, where $k$ is a value less than the number of columns of the matrix.

SVD is a commonly used linear algebra technique that factorizes a matrix into three matrices; a left unitary matrix, a diagonal singular values matrix, and a right unitary matrix. The formula for SVD is shown in equation 3.1.

$$M_{m \times n} = U_{m \times m} \Sigma_{m \times n} V_{n \times n}^T \tag{3.1}$$

The singular values matrix $\Sigma$ is often listed in descending order, which is important when using TruncSVD. In TruncSVD, only the $k$ columns of $U$ and $k$ rows of $V$ are calculated. These rows and columns should correspond to the $k$ largest singular values. TruncSVD thus relies on the truncated values being small enough for $M_{m \times n} \approx U_{m \times k} \Sigma_{k \times k} V_{k \times n}$ to be a good approximation. Using the obtained $U_{m \times k}$ to represent the matrix will finalize the reduction and made it dense, giving the truncated matrix the same number of rows as the original matrix.

### 3.2.2 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is an NLP technique for analyzing text documents and extracting useful data. The technique first uses term weights. In this case they have been calculated as a sparse tf-idf matrix of word weights. This matrix is transformed into a dense matrix through dimensionality reduction, in this case Truncated SVD. LSA works under the assumption that the distributional hypothesis holds; words that occur in similar contexts, such as documents, are inherently similar in meaning. In the case of this thesis, documents from the National Vulnerability Database (NVD) should possess a discernibly different context than Github issues. Therefore, the distributional hypothesis is assumed to hold for the purpose of this study.

### 3.2.3 T-Distributed Stochastic Neighbor Embedding

T-Distributed Stochastic Neighbor Embedding (t-SNE), is a dimensionality reduction technique commonly used to visualize high dimensional data.

T-SNE is used to plot and display the data clusters in a meaningful way. Figure 3.6 and Figure 3.7 uses t-SNE to properly display the clusters.

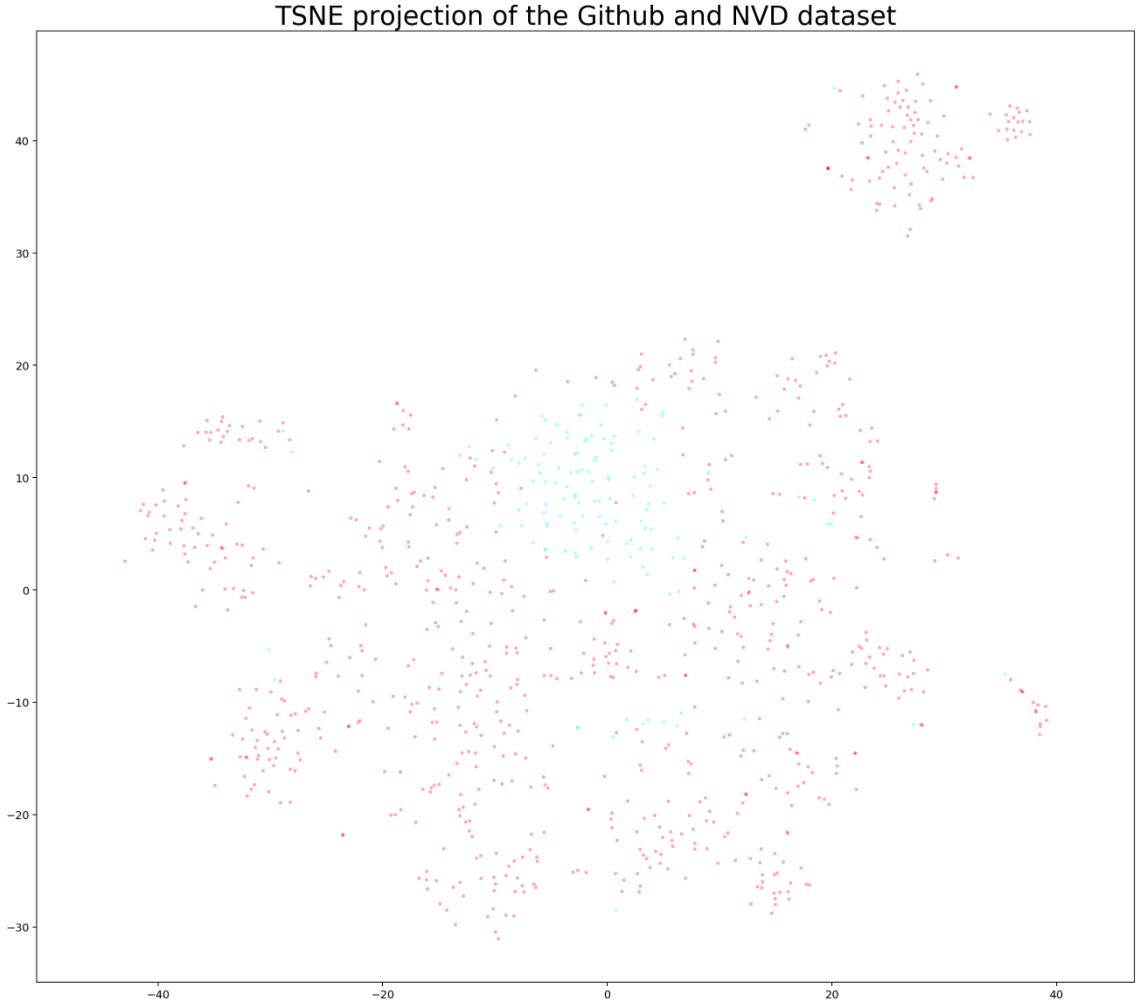


**Figure 3.6:** t-SNE plot showing the k-means clustered documents by Github and NVD source using tf-idf. Red dots correspond to Github issues and blue dots are NVD texts.

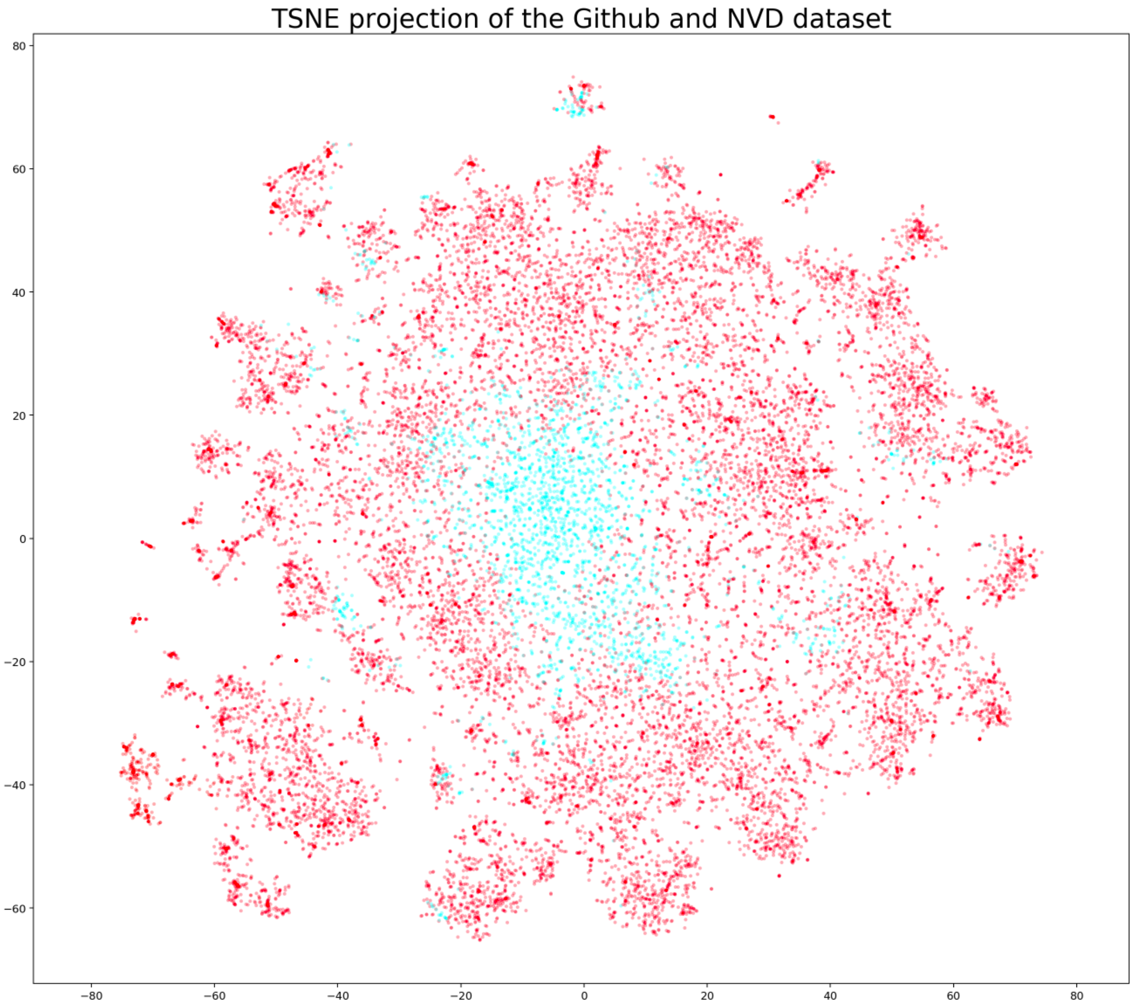


**Figure 3.7:** t-SNE plot showing the k-means clustered documents by Github and NVD source using tf-idf. Red dots correspond to Github issues and blue dots are NVD texts. Significantly more observations, greater than the other clustering plots for visualization purposes.

### 3.2.4 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) is a more recent dimensionality reduction technique that aims to optimize the mapping from a higher plane into two or three dimensions for visualization(McInnes et al., 2018). This method is still quite new and does not provide the same level of quality assurance when compared to a technique that has been in use for a longer period of time. McInnes et al. (2018) claim that UMAP is:

> demonstrably faster than t-SNE and provides better scaling.

This claim is inline with the observed calculations times for t-SNE and UMAP in this thesis, as can be seen in Figure 3.8. The observations are more closely clustered than in TSNE, which gives a better representation of the data.

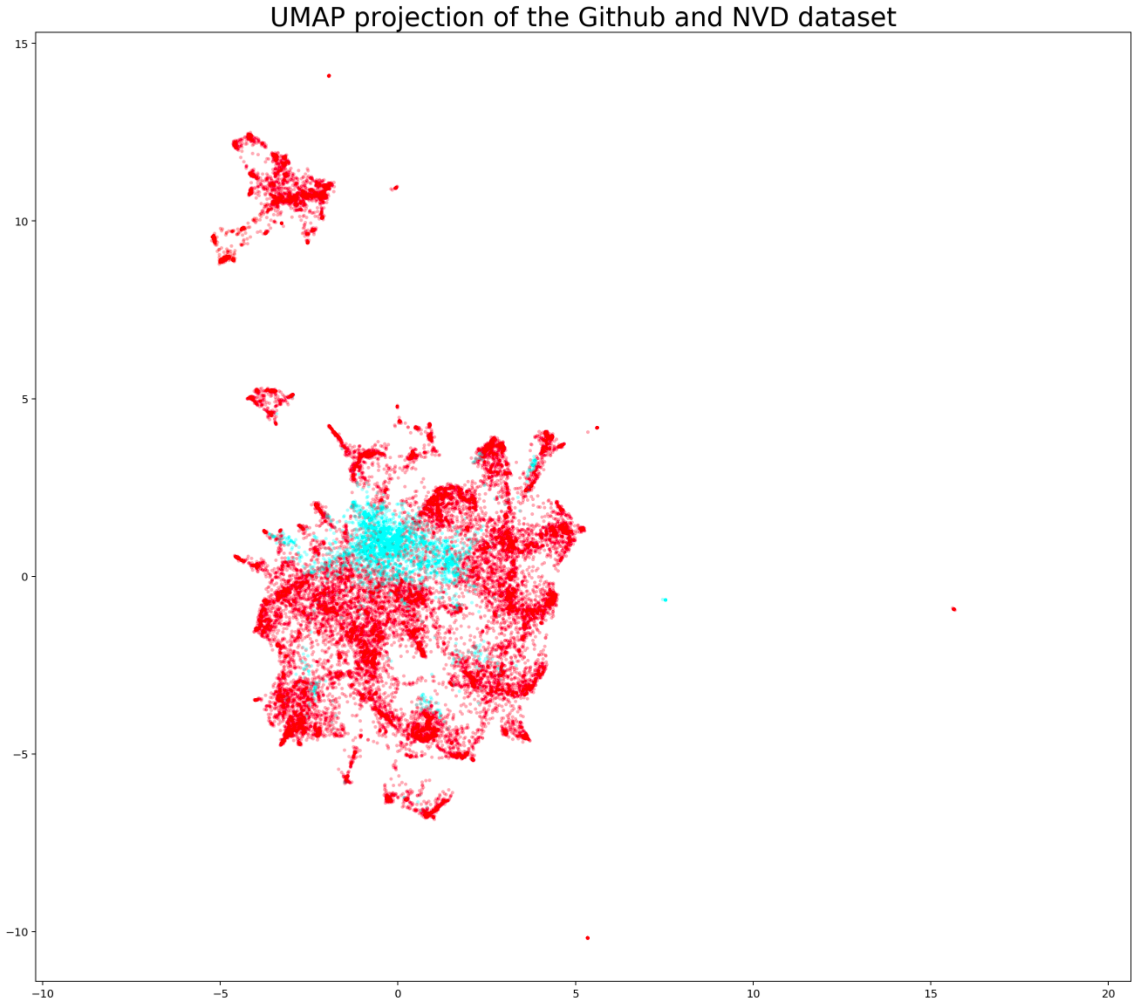
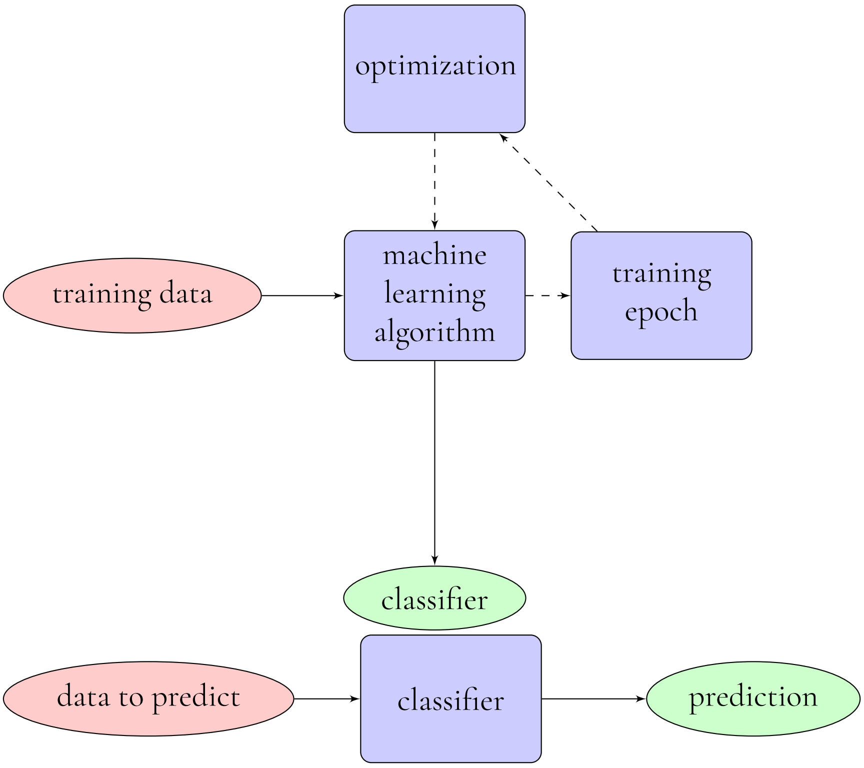


**Figure 3.8:** UMAP plot showing the k-means clustered documents by Github and NVD source using tf-idf. Red dots correspond to Github issues and blue dots are NVD texts.

# 3.3 Introduction to Machine Learning

Machine learning has been regarded as magic by the uninformed. This section aims to demystify the concept of machine learning and better explain the fundamental concepts required to understand academic work in machine learning. The core concepts that will be covered are: types of machine learning, overfitting and underfitting, batches and epochs, activation functions, optimization, and hyperparameters. In the figure below, the red nodes symbolize input to the system, blue is the system itself, and green is the output. The classifier is created using the machine learning algorithm and is a product of training. The classifier is then used in the following figure as an independent system which takes new data as input and outputs a prediction.



## 3.3.1 Supervised, Unsupervised, and Semi-supervised Learning

Supervised learning is the most common way to approach machine learning. Each observation in the training set contains both training data and a corresponding label. The model is then trained on these data-label pairs, making the model learn how to classify new observations without the label after the training. During training, the model updates its parameters based on the results.

Unsupervised learning on the other hand does not have access to any labels. It tries to learn from the data's internal structure. Example of common unsupervised learning methods are word embeddings, as explained in Section 3.1.2, and clustering, which is explained in the next subsection.

Semi-supervised learning tries to use a combination of supervised learning and unsupervised learning to make the model better, by making use of both labeled and unlabeled data during training. The reason why semi-supervised learning is interesting is because it is tedious to label data and there exists a lot of unlabeled data freely available on the internet.

### 3.3.2 Clustering

The core principle of clustering is to group observations into separate categories. Clustering can be useful for finding patterns or groupings that a human would not normally find through more intuitive approaches of categorization. There are various ways of clustering observations. One of the most common forms of clustering in data mining is the simple k-means clustering approach. K-means clustering is determined through setting $k$ cluster centers and then calculating the nearest cluster for each observation. The nearest cluster is the cluster center whose mean (from the observation) has the least squared Euclidean distance.

When the clusters have formed, each cluster has its center recalculated as the center of all of its observations. Each point is then reassigned to the nearest cluster (not necessarily the same as last iteration). This process continues either until a certain number of iterations have passed and may or may not converge. There is no guarantee for the convergence to reach a global optimum and as such, results may vary depending on initial cluster center allocation. Each observation is assigned to the cluster with the least squared Euclidean distance mean, that is the cluster whose points are closest on average to the observation to assign to a cluster.

### 3.3.3 Overfit and Underfit

A machine learning model is tasked with learning from the input data available to it. The patterns the model constructs to describe the data can overfit or underfit. Overfitting occurs when the model learns very complex patterns in order to perfectly fit the training data. This results in a model that will perform very well on the training data, but will fail to generalize to new and unseen data. Overfitted models have high variance, meaning that small differences in data will yield widely different results because the model has not learned the overarching patterns in the data and instead learns random noise. In contrast, the model can also underfit the training data, meaning that it learns too little from the training data. This results in high bias, making broad erroneous assumptions about the data by learning simplistic patterns. The trade-off in bias and variance of a model decides the ability to generalize to new data as well as the complexity of patterns learned. A method called dropout is commonly used to reduce overfit.

### 3.3.4 Batches and Epochs

When using a dataset in a neural network model, it is often a good practice to split the dataset into smaller batches. A batch contains a fixed amount of observations, usually chosen as a power of 2. The last batch of a set may be unbalanced.

Passing an entire dataset forward and backward through a network once is called an epoch. During training, multiple epochs are usually performed.

### 3.3.5 Gold Standard

Ideally, a ground truth should be used for evaluation of a machine learning model. Ground truth is the absolute truth, which will rarely be observable information. A gold standard is a dataset which aims to represent the underlying ground truth as accurately as possible. In the case of this thesis, the gold standard has been labelled manually by humans with some expertise in the field of computer security and will be assumed to be correct for proper evaluation. The main purpose of the gold standard is to ensure a high degree of certainty that a classifier's evaluation can be trusted. Ground truth and gold standard are often used interchangeably in the machine learning field, but will be referred to as gold standard below.

### 3.3.6 Activation Function

An activation function in the context of neural networks, is the function each node has that takes the inputs to the node and calculates the output from the node. The purpose of the activation function is to introduce non-linear behaviour. The choice of activation function can greatly impact the way a neural network works. The following activation functions are used in this study.

#### Rectified Linear Unit

Rectified Linear Unit or ReLU is a function that is zero for all negative input values and linear for all zero and positive values as seen in Figure 3.9, meaning that the activation is sparse. With fewer neurons sending a non-zero output, the network is more lightweight and less computationally expensive. The function is also computationally cheap and converges quickly as the function does not taper off at large input values. This means it will not suffer from the vanishing gradient problem.

$$f_{ReLU}(x) = \max(0, x) \tag{3.2}$$

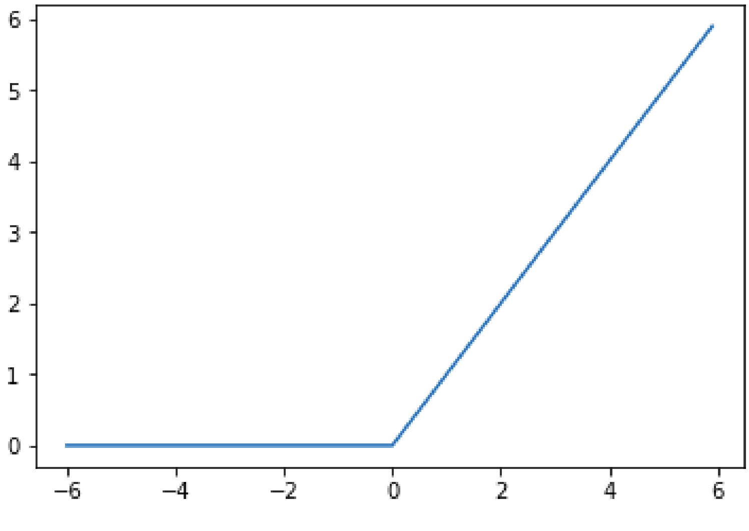


**Figure 3.9:** ReLU function

### Softmax Function

The softmax function is also called the normalized exponential function. The function takes a vector of real numbers and as the name suggests, normalizes them so the sum of the vector is 1. The vector then represents a probability distribution, proving quite useful when outputting a prediction from a multiclass classification problem.

The input vector $z$ has length $K$:

$$\begin{gathered}\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \\ i = 1, ..., K \\ z = (z_1, ..., z_K) \in \mathbb{R}^K\end{gathered} \tag{3.3}$$

The probability distribution has sum of 1 meaning that the probability vector covers all outcomes:

$$\sum_{i=1}^{K} \sigma(z)_i = 1 \tag{3.4}$$

### Sigmoid Function

The sigmoid function is bounded, meaning that the maximum and minimum y values are finite. It also only has positive derivatives at every point, giving it a characteristic sigmoid curve shape seen in Figure 3.10. Sigmoid functions are common in binary classification problems as a final layer to get a binary output. There are many sigmoid functions, the one used in this thesis is the logistic function, having the following formula:

$$f_{Sigmoid}(x) = \frac{1}{(1 + e^{-x})} \tag{3.5}$$

## 3.3.7 Backpropagation

Backpropagation (BP) is a commonly used algorithm during training in machine learning. It uses the weights of the model to efficiently compute the gradient of the loss function for a single sample. The algorithm works by calculating the gradient of the loss function. It does this in respect of each layer's weight using the chain rule, iteratively going backwards from the end layer-wise. This is an efficient way to calculate multi-variable derivatives.

## 3.3.8 Evaluation Metrics

Evaluation of model predictions is first measured and divided into true positives, false positives, true negatives and false negatives. True positives $t_p$ is the category of positive predictions that are actually from the positive class. False positives $f_p$ are incorrectly predicted as

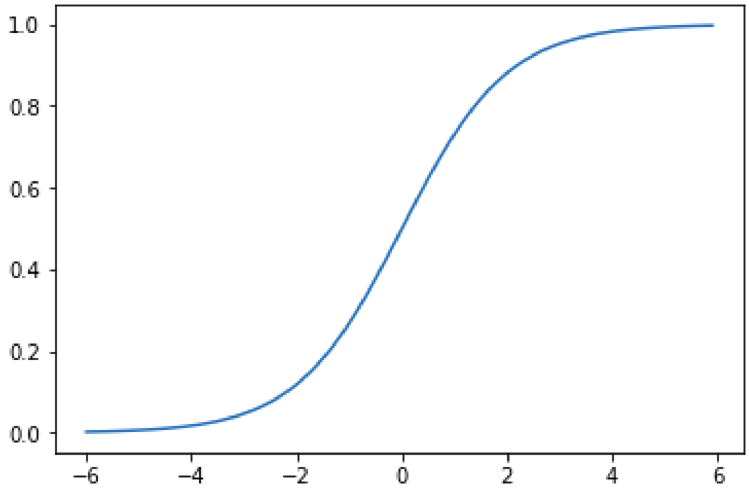


**Figure 3.10:** Sigmoid function

the positive class but is actually an element of the negative class. In the same vein, true negatives $t_n$ are negative predictions that are correct and false negatives $f_n$ that are incorrectly predicted as negatives but are from the positive class. Precision and recall is explained in Figure 3.11.

Precision is the measurement of correct predictions compared to the total predictions.

$$Precision = \frac{t_p}{t_p + f_p}$$

Recall is measured as the detected elements of the class in proportion to the total scope of the class.

$$Recall = \frac{t_p}{t_p + f_n}$$

F1 score can be calculated with different formulae, the following formula expresses the traditional F1 score function that was used in this thesis, calculating the harmonic mean of precision and recall.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

# 3.4 Optimization

## 3.4.1 Stochastic Gradient Descent

Gradient descent is defined as the minimization of the objective function $f(\theta)$ where $\theta$ is the model's parameters. The gradient is calculated at each iterative step and the parameter $\theta$ is updated in the opposite direction of the gradient by an amount based on the learning rate.

The learning rate controls the scale of updates to the weights. A lower learning rate value leads to smaller weight changes and slower convergence towards the optimum. A higher

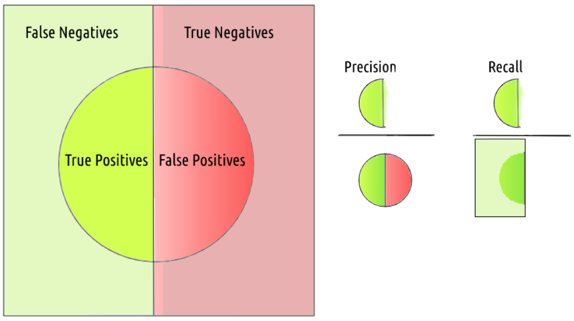


**Figure 3.11:** The left side are all instances of the positive class, on the right are the negative instances. The circle shows correct predictions and the outer rectangle shows the incorrect predictions. Precision is the proportion of true positives out of the total positive predictions. Recall is the proportion of positives found out of the total number of positive instances. As such, a high precision reduces false positive rate and increases true positive rate. High recall improves the total scope of positives found.

learning rate converges faster, but at a greater risk of overshooting the target and in the worst case not converging at all. The intention in gradient descent is to reach the global minimum. There are several issues that can arise in gradient descent, such as getting stuck in a local minimum during optimization. If the learning rate is too high, it is possible that the algorithm will not converge to a minimum. In contrast, a low learning rate leads to slow optimization and risk of underfitting.

In machine learning, stochastic gradient descent (SGD) is primarily used. It is a stochastic approximation of gradient descent, replacing the gradient with an estimation of it. In SGD, the gradient is calculated using a random subset of the data, instead of using the entire dataset. Backpropagation is used to efficiently compute this gradient.

There are many SGD optimization algorithms and some popular algorithms will be mentioned in this section. For further reading, refer to the gradient descent optimization overview by Ruder (2016).

The Adaptive Gradient algorithm (AdaGrad) (Duchi et al., 2011) has the learning rate adjusted for each parameter. Infrequent parameters have a higher learning rate for more substantial updates. Frequent parameters instead have lower learning rate, leading to smaller updates but more frequent iteration. This method achieves good performance on sparse gradients such as nlp tasks(Ruder, 2016).

Root Mean Square Propagation (RMSProp) similarly to AdaGrad, has per-parameter learning rates. The learning rates are adjusted based on the first moment or mean gradient.

### Adam

The optimizer primarily used in this thesis is the Adam optimizer proposed by Kingma and Ba (2014). Adam is short for adaptive moment estimation, building on the fundamentals of AdaGrad and RMSProp. In Adam, the optimizer calculates the mean gradient like in RMSProp and additionally the second central moment or variance gradient. The combination of these two calculations are used to change the parameter learning rates. The exponentially decaying averages of the first ($g_t$) and second ($g_t^2$) moment of the gradients from previous iterations are calculated as following:

$$\begin{aligned} m_t &= \beta_1 * m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{3.6}$$

$m$ is the mean (first moment) and $v$ is the uncentered variance (second moment). $\beta$ is the decay rate for each equation. $\beta$ close to 1 corresponds to very slow decay.

There is bias-correction that accounts for a bias towards zero for the $m$ and $v$ vectors as they are initialized as zeroes. The first and second moment are estimated to:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{3.7}$$

The updated parameters $\theta_{t+1}$ are derived from the following equation utilizing the first and second moment in addition to the learning rate $\eta$ and the smoothing term $\epsilon$:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.8}$$

### 3.4.2 Hyperparameters

Parameters used in machine learning can be divided into two categories: mutable and immutable. The property describes the parameters' ability to change during training.

Parameters are either derived during training or set in advance. The ones specified before training begins are called hyperparameters. Some hyperparameters may also be mutable. Common hyperparameters for neural networks are learning rate, batch size, number of epochs, and number of cells in each layer. Learning rate is typically set to a certain value before training and in some cases uses learning rate decay with each epoch during training. This results in the model quickly adapting during the early stages of training followed by a more controlled convergence towards the optimum.

# Chapter 4

# Data

In machine learning tasks, the data is an essential part of the problem statement. In the case of vulnerability detection in text, there are several questions that must be answered before considering the usage of NLP. The entire research process is documented and divided into sections, including the Data chapter and Models chapter. Data Acquisition describes how the data was gathered. Next, some of the data is annotated for future use in Data Annotation. Exploratory Data Analysis refers to the practice of learning useful patterns in the data.

- Data acquisition: is it possible to gather data of sufficiently large quantities to effectively use machine learning?

- Data annotation: what restrictions limit data annotations and what annotation guidelines should be used?

- Data cleaning: what information is important in the data and what should be filtered out?

- Exploratory Data Analysis: do patterns exist in the data? Can the problem statement be answered with the type of information available?

Examples of data samples can be found in the Introduction in Table 1.1 as well as in the Appendix Example 9.3.

## 4.1 Data Acquisition

Our unlabeled data is scraped from Github (Github, 2020) and National Vulnerability Database (NIST, 2020). The Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE) descriptions from the National Vulnerability Database (NVD) can safely be considered security related.

| Dataset | Github | NVD | Gitlab |
|---|---|---|---|
| **Train Dataset** | | | |
| non-security related | 47095 | 0 | 460 |
| security | 3691 | 47662 | 452 |
| **Validation Dataset** | | | |
| non-security related | 4683 | 0 | 66 |
| security | 453 | 5242 | 55 |
| **User Labeled Test Dataset** | | | |
| non-security related | 555 | 0 | 0 |
| security | 514 | 0 | 0 |
| **Debricked Labeled Test Dataset** | | | |
| non-security related | 835 | 0 | 0 |
| security | 112 | 0 | 0 |

**Table 4.1:** Table presents the distributions of data from different sources by class.

The data from Github consisted of publicly posted issues from popular repositories. The issues were often user submitted and described the topic with varying degrees of precision and with differing levels of comprehension of the English language. Some issues were not in English. The issue data could be considered highly variant overall. The data from NVD in contrast to the Github data, was incredibly consistent in vocabulary, overall language, and format. Note that these descriptions are quite different from issue descriptions on Github. The differences in these texts were to be evaluated in the following section, which deals with providing a better understanding of the data. A substantial labeled dataset, User Labeled Test Set, from the SRN paper is used(Palacio et al., 2019). This set was generated through combining NVD data with Github and Gitlab issues labeled as security related or not. Note that an overwhelming majority of security related data is from NVD.

Since there is a risk that the model is trained to predict if a text comes from Github, Gitlab, or NVD instead of if the content is security related, the test sets used contain only Github data. More security issues from other sources than NVD could improve the training results as the domains will be more similar in regards to testing and training.

## 4.2 Data Annotation

Proper evaluation of the models requires labeled data to test against. Firstly, the SRN dataset is split into train, validation, and test sets. The test set, as previously mentioned, only contains data from Github.

While these sets should be sufficient, we personally annotated over 1000 Github issues to have a gold standard to test against. We discovered that few issues on Github are actually security related, around 1% were actual vulnerability reports. We settled on creating slightly different annotation guidelines that valued potential security risks as security related. This came to include for example issues about crashes and memory leaks. Since this problem statement or test set is quite different from the training and validation data, we expect these results to be significantly worse than our other test set, derived from the same annotation guidelines as the training data.

Manual human labeling was required to create a gold standard. Instructions on how to annotate were specified to keep the annotations consistent with several annotators. Refer to section 9.1 for details. The annotation policy has five categories with an ascending associated risk for each category. The highest risk is the Vuln category which contains known exploits and user reported vulnerabilities. The next category, Risk, contains memory leaks, unrestricted user inputs, access violations among others. In the safest categories, the subject matter covers for example design and questions unrelated to code.

In order to address the issue of few security related texts, different methods of sampling from the unlabeled dataset were used. The first 300 entries were extracted using uniform distribution sampling. The next method of sampling used the document similarity scoring method found in Subsection 4.4.3. Lack of labeled and categorized data necessitated this method, but note that it is biased.

Annotating whether a text is about security was not always straightforward, since it requires more domain specific understanding of the meaning of the issue. For example, the problem of annotating if a text is positive, negative, or neutral should be a much easier task and as such, result in high annotation similarity. Having established that the problem was difficult to annotate for the two annotators, this is a source of potentially inaccurate data for the model. When annotations were made for the same data, the annotations were compared and discussed. Later on, this process was automated and the higher risk annotation value was chosen when conflicting annotations were made.

## 4.3 Data Cleaning

After accumulating the labeled data that is needed, the next step is cleaning the data. In order to properly read the data, it needs to be tokenized. Tokenization is a process that splits the text strings into tokens, with the resulting tokens being for example words and punctuation. Without cleaning the data first, it would be difficult to know where the string should be split. The primary focus of data cleaning should be to allow for as useful tokens as possible.

- Words that are connected to punctuation should still end up as the correct base word. Example: "word." should be split into "word" and ".".

- Non-English text: The model we are building will not be trained to understand any other languages than English and will only use English embeddings, therefore, we discard all documents that contain non-English characters, such as Cyrillic script or kanji.

- Documents that contain only a few words or too many words are removed as they are deemed to not contain important information. There is a lower limit to how useful a few words can be. The lack of substance in the outliers was empirically evident and they were removed from the training data.

- Code segments were removed in the capacity that was possible, but it is possible that other models are able to take advantage of this type of text. This aspect was considered outside the scope of this study.

## 4.4 Exploratory Data Analysis

With machine learning problems, it is essential to understand the training data used to learn to solve the problem. The techniques that were utilized in this step include clustering, plotting the clusters utilizing dimensionality reduction, n-gram counting, and tf-idf scoring.

### 4.4.1 Distributions

The Github data was first uniformly sampled and annotated for the purpose of understanding the data. Unbiased sampling may help to understand the distributions of various data types. From the issues that were annotated, it was observed that staggeringly few observations were even vaguely related to computer security. With this in mind, the definition of security related text was initially decided to be somewhat lenient and inclusive. The issue of unbalanced data distributions will be elaborated upon in the Discussion section. The efforts to cluster the data with t-SNE and UMAP indicated that the Github and NVD datasets were decidedly different. Plot 3.6 and plot 3.8 show that NVD and Github observations are mixed. Ideally, the security related Github issues would all be clustered with various NVD dominant clusters and the safe issues would be completely separated. Most common words in these clusters can be seen in the appendix 9.4.

A variety of biased sampling methods were tried in order to receive more balanced distributions. Metadata and features were extracted from NVD data in order to find meaningful descriptors for computer security. This was accomplished by incorporating top word n-grams extraction and calculating tf-idf vectors to learn word weights for computer security related contexts. With these features, the biased sampling was possible.

### 4.4.2 N-Grams

Unigrams, bigrams, and trigrams were extracted from two distinct sources: Github and NVD. The n-grams from these sources were extracted both from the raw sources and cleaned sources. The n-gram sets were compared to find patterns in the language used in these sources, as seen in tables 4.2 and 4.3. Complete lists of top n-grams can be found in the Appendix 9.2. After comparing the two sources, the common n-grams in Github issues that are not common in NVD were removed from NVD n-grams. The goal is to filter NVD n-grams to only contain security related n-grams. The NVD security n-grams filter the Github issues and remove any issues not containing security n-grams. The result was a dataset with a high degree of vaguely security related issues. This process creates insight into the data that will be learned from in the training stage. The n-gram filtered dataset can be used at later stages as training data if it is of high quality, which can be ascertained by manually checking a uniformly sampled subset.

**Top Word Unigrams Descending Order**

| NVD (filtered) | Github |
|---|---|
| allows | js |
| vulnerability | error |
| attackers | node |
| improper | version |
| arbitrary | file |
| cve | com |
| web | lib |
| site | using |
| execute | use |

**Table 4.2:** Unigrams: single terms with no spaces.

**Top Word Bigrams Descending Order**

| NVD (filtered) | Github |
|---|---|
| remote attackers | node modules |
| allows remote | github com |
| cross site | youtube dl |
| execute arbitrary | usr lib |
| cve cve | py line |
| denial service | usr local |
| site scripting | https github |
| cause denial | steps reproduce |
| attackers execute | framework versions |

**Table 4.3:** Bigrams: pairs of terms separated by a space.

### 4.4.3 Document Similarity Scoring

One sampling method that was attempted was tf-idf document source scoring. Previous work could not be found in academic papers, but it was considered an interesting experimental approach for ranking the relevance of a document. Tf-idf scoring firstly calculates tf-idf vectors on the corpus corresponding to each data source and normalizes the vectors using the L1-norm. The averaged sums of the tf-idf vectors produce an averaged tf-idf vector. Each issue from Github is then scored with each of these vectors and the tf-idf vector that produces the highest score is chosen as the issues' source. The issues that were predicted to derive from NVD but were actually from Github were considered interesting and sampled out. The documents with a score lower than the median were discarded as being irrelevant and scores that were too similar between the Github score and NVD score were also discarded. The NVD tf-idf score as such had to be distinctly higher than the corresponding Github tf-idf score. The tf-idf score describes the amount of corpus specific terminology the text contains, which enabled finding documents that are as unique as possible. These samples were found to contain a substantially higher proportion of security related issues.

# Chapter 5
# Models

The Baseline section establishes a simple initial document classifier model to see if the problem statement seems solvable with NLP. Following the baseline implementation, more complex models are constructed in neural networks. Finally, the evaluation process for model comparisons is described.

## 5.1 Baseline

It is pervasive within machine learning to create a simple baseline early in the development phase in order to form some initial assessments about the problem's nature. The baseline model should primarily be used to explore how difficult the chosen problem. The baseline will also provide a base for comparison with more complex architectures.

### 5.1.1 Logistic Regression

A binary logistic regression classifier on tf-idf vectors was chosen in order to establish what a basic model could achieve in terms of classification strength. Later on, the more complex models will be compared to this classifier in order to gain context as to how it performs. A neural network will often perform better than a logistic regression classifier, but it cannot be assumed to be true.

### 5.1.2 Silver Standard

The data annotations needed for the project is difficult to outsource to other annotators as expertise in the computer security domain is required. It was quickly ascertained that a silver standard of high quality is essential to compensate for the lack of outsourcing. A logistic regression classifier was trained on a subset of the gold standard found in Subsection

3.3.5 and evaluated on another subset. The classifications demand a high degree of certainty; probability scores above 95% or below 5% were chosen. It was deemed that 5 percent data uncertainty was low enough that the mislabeled data will largely be ignored or not have a large impact on the training. These silver observations are then added to the training pool together with a small subset of NVD-data labeled as security-related. The model is then retrained using the new training pool as its training input. This iterative process improves the model slowly while building a silver standard.

The silver standard generated through the logistic regression pseudo labeling was not used to train the neural network in the end. The gold standard training data used to acquire the silver standard could not be used for testing as it was biased and had been seen by the logistic regression model. In the end, a larger test set was prioritized over a silver standard training set in order to improve confidence in the evaluations.

The silver standard generated through the use of issue tags and NVD data also possesses some bias since it is in part derived from user reported vulnerabilities and does not contain unreported vulnerabilities.

## 5.2 Model Architectures

We intend to further expand on security text classification with a different NLP approach, specifically the Hierarchical Attention Network (HAN) architecture built on RNNs and attention mechanisms. While the problem statement is similar to the previously discussed SRN study (Palacio et al., 2019), the purpose is to explore alternative solutions to this problem, evaluate on a proper gold standard annotated by us, and put the task into context through benchmarking. With an implementation of the SRN model at hand, benchmarking and proper evaluation can be found in the Results chapter 6. We also intend to lay some groundwork for SSL approaches. The Model Architectures section covers the theoretical basis for the neural networks implemented, specifically CNN, HAN, and VAT.

### 5.2.1 Convolutional Neural Network

Convolutional Neural Networks were initially developed for the computer vision domain. Like many other machine learning techniques, CNNs have been adapted for the text domain with great success. It has been shown to be effective on the text domain to a similar degree as LSTMs and GRUs(Lopez and Kalita, 2017)(Bai et al., 2018).

CNNs use a kernel to mask over the input data and output a single value at each step as seen in Figure 5.1. The weights of the kernel are used to calculate the output value. In the case of CNNs in NLP, the kernel size is typically limited to word n-grams (a number of words) by the number of embedding dimensions.

CNNs can be tricky to tune hyperparameters successfully, for more information on good practices refer to the article by Zhang and Wallace (2017).

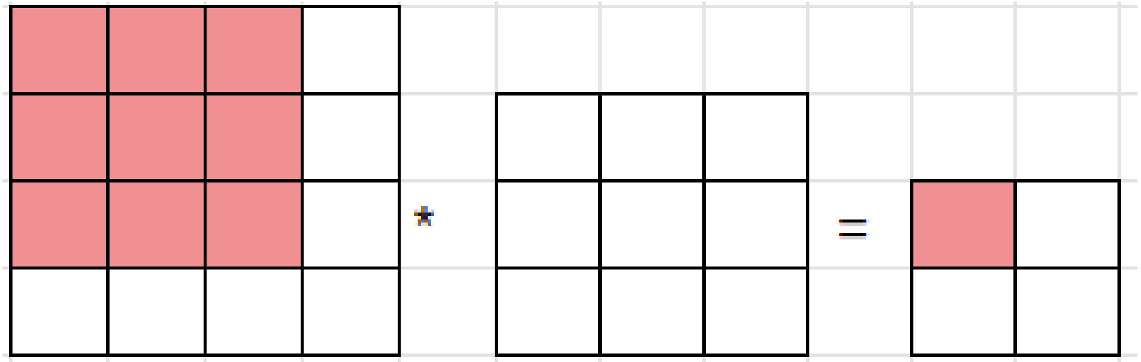


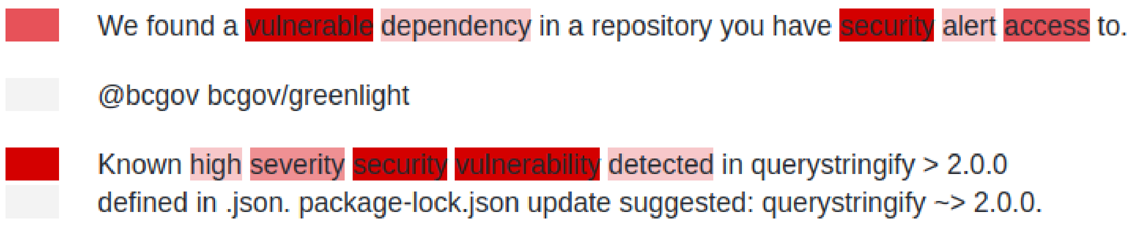
**Figure 5.1:** The figure shows how the kernel (in red) calculates one of the output cells (red highlight on the right grid). Kernel is size 3x3 meaning that with a step size of 1, there will be four steps in total. The output shape is therefore 2x2

### 5.2.2 Attention

Attention originated from the sequence-to-sequence modelling problem, such as machine translation, in the text domain. Previously, sequence-to-sequence problems were often solved by using an encoder and decoder on an input sequence and predicting a fixed length output sequence. An encoder is responsible for mapping the words of a sentence into a fixed length context vector in another space. The decoder receives the vector and maps it back to natural language space. The encoder and decoder are neural networks. The fixed length restriction in this approach was shown to decrease performance when used on longer sentences.

Attention in its first iteration (Bahdanau et al., 2014) predicts one word at a time while only looking at the subset of the input sequence with most perceived importance. Attention has an encoder and a decoder. The decoder takes a context vector for each word instead of per sentence. In this implementation, the attention layer is built with a bidirectional LSTM and therefore combines hidden states forward and backward.

A myriad of variants have been developed since attention's inception, including the self-learning variants, for example the Transformer architecture(Vaswani et al., 2017).



We found a vulnerable dependency in a repository you have security alert access to.

@bcgov bcgov/greenlight

Known high severity security vulnerability detected in querystringify > 2.0.0 defined in .json. package-lock.json update suggested: querystringify ~> 2.0.0.

**Figure 5.2:** Example of attention mechanism both for word level and sentence level attention. The red word highlights indicates relevance to the sentence. The red highlights to the left of each sentence shows the relevance of each sentence to the document. Grey or non-highlighted words are deemed irrelevant to the core message of the document.

### 5.2.3 Hierarchical Attention Network

Hierarchical Attention Network (HAN) for document classification was first introduced by Yang et al. (2016). The paper proposes a model based on a hierarchical structure that tries to mirror the structure of a document, by having one level focusing on the words and one level focusing on the sentences. The implementation of HAN used is based on the model described by Yang. A word encoder embeds the words into vectors, which are then passed on to an attention layer that extracts the most meaningful words for the sentence into a summarized sentence. The authors of the paper note that characters could be used to generate the word vectors as an additional level instead of directly using word embedding. The sentences go into a sentence encoder followed by a sentence level attention layer. The sentences build a succinct document vector representation. Both levels of the structure consist of one encoder and one attention layer. The output of the model, which is a document vector, then goes through a softmax layer to get a probability for the classification task. This structure can be viewed in Figure 5.3.

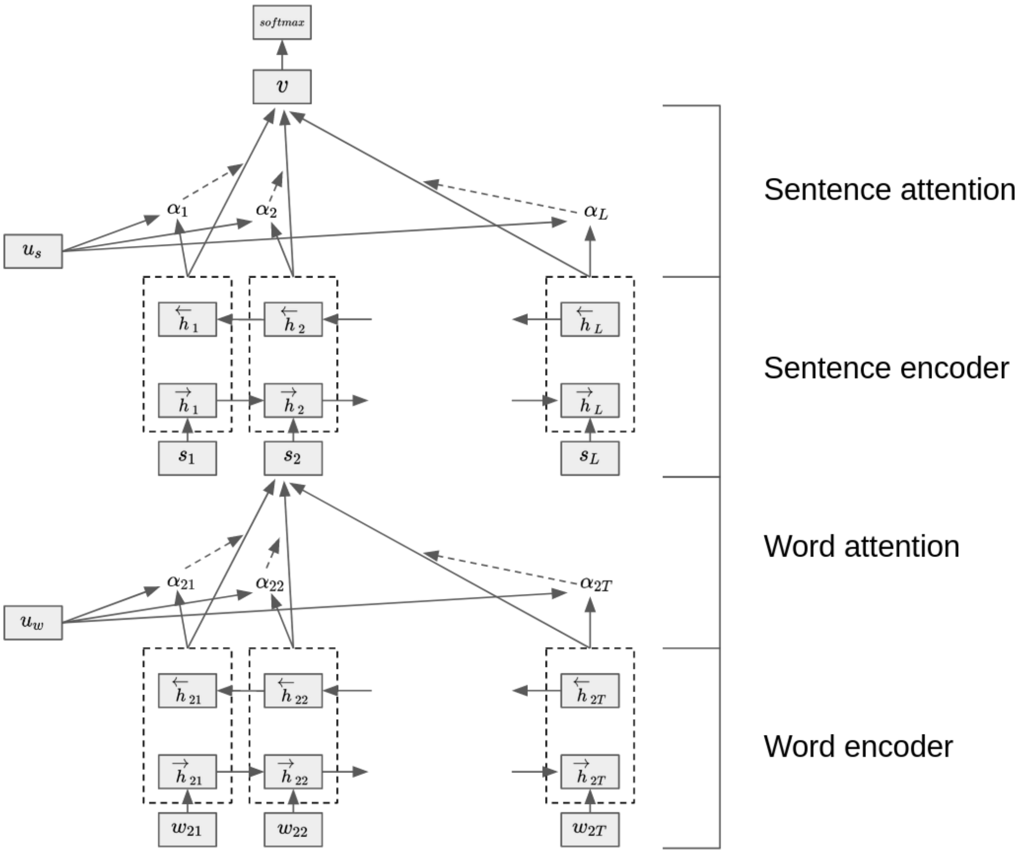


**Figure 5.3:** The structure of HAN.

The main model investigated in this thesis uses a HAN classifier, using LSTM as encoders and simple attention with context as its attention layers.

The first layer of the HAN architecture is the word encoder. Just like the first attention variant by Bahdanau in 2014, HAN uses a GRU sequence encoder. A GRU has two types of gates: the reset gate and update gate. The purpose of these gates is to modify the hidden state transition. The update gate controls what is kept and removed from the old state as well as what information to add when updating to the next state. The reset gate controls how much information from the previous state to forget(Nguyen, 2018).

Following the word sequence encoder, the output is passed into a word-level attention layer. For HAN, Yang et al. (2016) engineered attention with context to:

> discover when a sequence of tokens is relevant rather than simply filtering for (sequences of) tokens, taken out of context.

The word annotation $h_{it}$ is inputted into a one-layer multilayer perceptron with weight $W_w$ and bias $b_w$ to extract the corresponding hidden state $u_{it}$, using $\tanh$ as the activation function. The weight $\alpha_{it}$ is calculated with a word-level context vector $u_w$ attention scheme and is normalized with a softmax function. Lastly, a sentence vector $s_i$ is computed as a weighted sum of the word annotations and their calculated weights. Attention with context can be viewed in the following equation.

$$
\begin{aligned}
u_{it} &= \tanh\left(W_w h_{it} + b_w\right) \\
\alpha_{it} &= \frac{\exp(u_{it}^{\top} u_w)}{\sum_t \exp(u_{it}^{\top} u_w)} \\
s_i &= \sum_t \alpha_{it} h_{it}
\end{aligned}
\tag{5.1}
$$

It is possible to generalize this approach to character and sentence level attention as well. In the case of sentence attention, which is used in HAN, the final output is an concise document vector.

The document vector is used for document classification using a softmax function.

## 5.3 Semi-Supervised Learning

Most neural network models are using supervised learning, which are trained with already labeled data. For every data instance fed into the model during training, the data have a corresponding label attached to it. Semi-supervised models differ in that in addition to the labeled observations, they try to take advantage of unlabeled data as well.

The main semi-supervised learning approach tried in this thesis is Virtual Adversarial Training (VAT). VAT is a regularizing method modifying the loss-function, making it deployable in an existing model. To better understand VAT, basic Adversarial Training (AT) is first explained.

### 5.3.1 Adversarial Training

Adversarial Training is a supervised method based upon creating adversarial examples. It was first introduced by Goodfellow et al. (2014b). The adversarial examples are created by modifying existing examples with a small perturbation in a direction that makes the model misclassify the adversarial example with as high degree as possible. The idea behind the method is to use observations that are very close in input space, but very far away from each other in the model output space. If these points exists and the model has not trained with adversarial examples, then there exist small perturbations that will make the classifier misclassify by adding the perturbation to the example. By letting a model train on these adversarial examples the model can learn to regularize and generalize better. These perturbations are often too small for a human to notice.

Adversarial Training modifies only the loss function, making it applicable on already existing models. Denote $x$ as the input, $y$ as the label paired with $x$, $\theta$ as the parameters of the model, $\hat{\theta}$ as the parameters with a backpropagation stop, and $r$ as a small uniformly sampled perturbation with the same dimension as $x$. The $\epsilon$ is a hyperparameter that restricts the absolute value of $r$. The adversarial loss $L_{adv}$ can then be viewed in equation 5.2. Stopping the backpropagation in $\hat{\theta}$ means that the backpropagation algorithm should not be used to propagate the gradients in the case of $\hat{\theta}$.

$$\begin{aligned} L_{adv}(\theta) &= -\log p(y|x + r_{adv}; \theta) \\ r_{adv} &= \arg \min_{r, \|r\| \leq \epsilon} \log p(y|x + r; \hat{\theta}) \end{aligned} \tag{5.2}$$

## 5.3.2 Virtual Adversarial Training

Virtual Adversarial Training (VAT) is an extension on Adversarial Training making it accessible in a semi-supervised environment(Miyato et al., 2015). It works similar to Adversarial Training, but instead of using the labels to determine how the perturbation should be created, it tries to follow the direction of the gradient using an approximation. This is done by calculating the Kullback-Leibler divergence ($D_{KL}$) between the probability distribution of the input and the probability distribution of the input plus a small random perturbation.

The $D_{KL}$ between two discrete probability distributions $P$ and $Q$ on the same probability space $\chi$ is defined as

$$D_{KL}[P||Q] = \sum_{x \in \chi} P(x) log(\frac{P(x)}{Q(x)}$$

The VAT cost is calculated using the equation 5.3, using the same variables as denoted in Adversarial Training with the addition of $D_{KL}$ as the Kullback-Leibler divergence.

$$\begin{aligned} L_{v-adv}(\theta) &= D_{KL}[p(\cdot|x, \hat{\theta}) \| p(\cdot|x + r_{v-adv}; \theta)] \\ r_{v-adv} &= \arg \max_{r, \|r\| \leq \epsilon} D_{KL}[p(\cdot|x; \hat{\theta}) \| p(\cdot|x + r; \hat{\theta})] \end{aligned} \tag{5.3}$$

In the equation, the probability distributions are denoted as placeholder distributions, $p(\cdot| \ldots)$. The actual distribution used will vary depending on the problem.

A classifier is trained to be smooth by minimizing equation 5.3, which can be considered to making the classifier resilient to worst-case perturbation(Miyato et al., 2015).

## VAT in Text Classification

VAT in text classification was first proposed by Miyato et al. (2016). It expands VAT into the text domain. Since text basically is a sequence of words, the algorithm needs to be updated to handle sequences instead of just raw input.

Denote $s$ to be a sequence containing word embeddings, $s = [\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_k]$ where $\hat{v}_i$ is a normalized word embedding using the equation 5.4.

$$\begin{aligned} \hat{v}_i &= \frac{v_i - E(v}{\sqrt{Var(v)}} \\ E(v) &= \sum_{j=1}^{K} f_j v_j, Var(v) = \sum_{j=1}^{K} f_j (v_j - E(v))^2 \end{aligned} \tag{5.4}$$

The word embeddings need to be normalized to avoid making the perturbations insignificant by learning embeddings with very large norm. In equation 5.4, $E$ is the expectation and $Var$ is the variance.

In Adversarial Training for text classification, the updated loss function for sequences can be seen in equation 5.5. The variables are used in the same way as previous subsections, as in equation 5.2 and in equation 5.3, but with the addition of $\nabla$ being the gradient calculated efficiently during backpropagation and $N$ being the number of labeled entries in the dataset. The symbol $\nabla_x$ is the gradient using the observation $x$ during backpropagation. Figure 5.4 illustrates embedding perturbation as is used in VAT on text.

$$\begin{aligned} L_{adv}(\theta) &= -\frac{1}{N} \sum_{n=1}^{N} \log p(y_n | s_n + r_{adv,n}; \theta) \\ r_{adv} &= -\epsilon g / ||g||_2 \\ g &= \nabla_s \log p(y|s; \hat{\theta}) \end{aligned} \tag{5.5}$$

By using a sequence of word embeddings as the input instead of the sequence of the tokenized words, applying the perturbations obtained from the VAT-calculation directly on the embeddings will create adversarial examples suitable for text, as shown in figure 5.5.

In VAT for text classification the approximated virtual adversarial perturbation is calculated using the equations in Equation 5.6. This is done at each training step. The number of labeled and unlabeled examples are denoted as $N'$, but otherwise the same variables are used

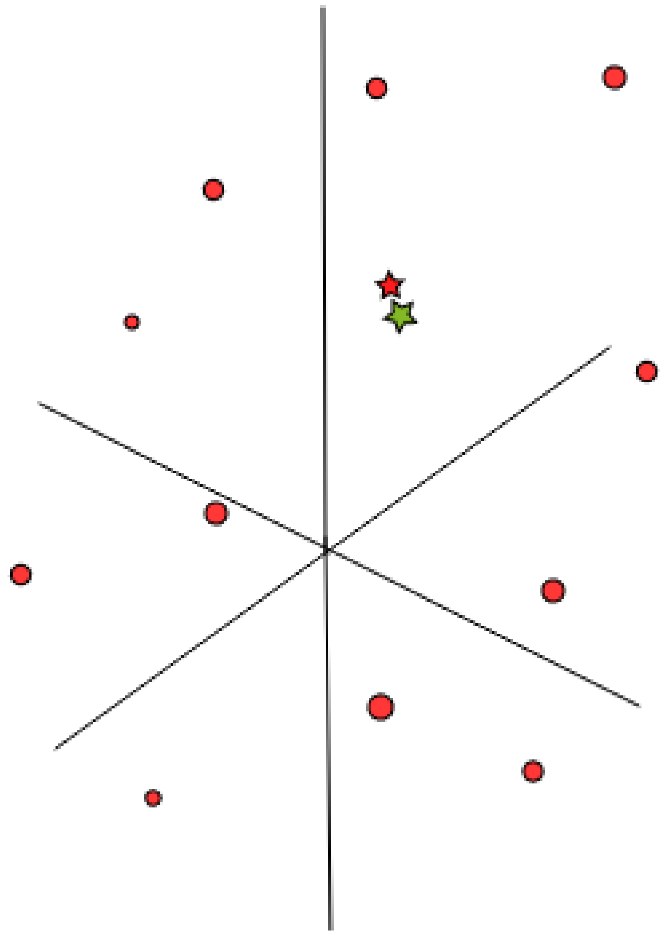


**Figure 5.4:** VAT perturbation of the embedding values for the word represented by the red star into the value represented by the green star. The embedding value is still very similar, but the value in value space no longer necessarily matches any word in word space. The red dots represent some other arbitrary words as embedded values in 3d vector space.

as in equation 5.2, Equation 5.3 and in Equation 5.5.

$$\begin{aligned}
L_{v-adv}(\theta) &= \frac{1}{N'}\sum_{n'=1}^{N'} D_{KL}[p(\cdot|s_{n'};\hat{\theta})||p(\cdot|s_{n'}+r_{v-adv,n'};\theta)] \\
r_{v-adv} &= \epsilon g/||g||_2 \\
g &= \nabla_{s+d} D_{KL}[p(\cdot|s;\hat{\theta})||p(\cdot|s+d;\hat{\theta})]
\end{aligned} \tag{5.6}$$

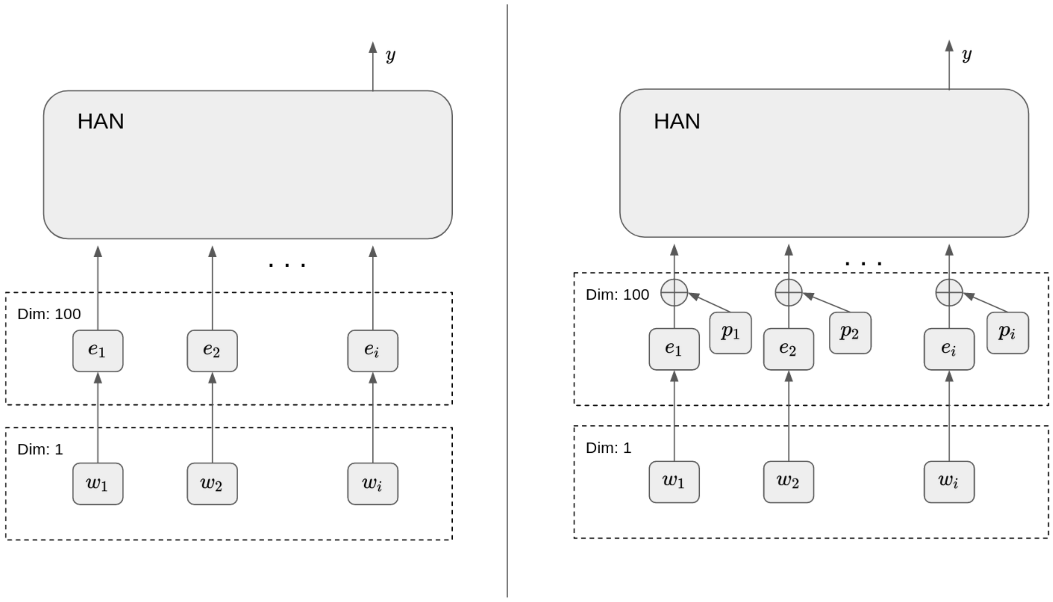


**Figure 5.5:** Left picture shows a simplified picture of embeddings with HAN, right picture shows the perturbed embeddings with HAN. Dim shows the output dimension of each layer and y is the output of the network.

## 5.4 Neural Networks

After establishing a simple baseline Logistic Regression, the results suggested that the problem could be solved with machine learning. At this point, more complex model architectures were considered. There are different advantages to recurrent neural networks (RNNs) and convolutional neural networks, as stated in Yin et al. (2017). Previous work use CNNs in the context of security text classification (Palacio et al., 2019).

We chose to implement a HAN model utilizing a RNN layer. This was in part because a recent study, which proposed the SRN model, had already established that CNNs were effective in this classification domain at this time. The study only compared against variations of itself and did not leave test data to allow benchmarking, we found there was room to further explore both the potential of CNNs and RNNs in this task. The CNN model is a publicly available implementation of SRN made by the authors, which only requires some extra lines of code to make work. The model itself is there in its entirety but the hyperparameters are not tuned the same as their private versions. In this thesis, we aim to do the SRN model justice with our own hyperparameters and benchmark against the same testsets for both our HAN model and our version of SRN.

### 5.4.1 Hierarchical Attention Network

The HAN architecture consists of a word level section followed by a sentence level section. The model can be seen in Figure 5.6. The input to the model is the text document data. The first layer is a frozen embedding layer, mapping each word to the corresponding stored embedding values. This is followed by a spatial dropout layer, first proposed by Tompson et al. (2014), which randomly discards a fraction of the words in each input text. This method has previously been shown to reduce overfitting. The model also makes use of normal dropout, helping reduce overfitting by randomly dropping output from a fraction of the neural network's cells.

The LSTM is a CuDNNLSTM optimized for Nvidia GPUs for quicker training, which leaves more room for hyperparameter tuning. The next layer is attention with context at a word level. The attention layer keeps only the most important words of each sentence in the document text. The word encoder model described above is input to a time distributed layer along with sentence divided document text.

A bidirectional LSTM at a sentence level is followed by attention with context on a sentence level, meaning that the most relevant sentences of each document will remain.

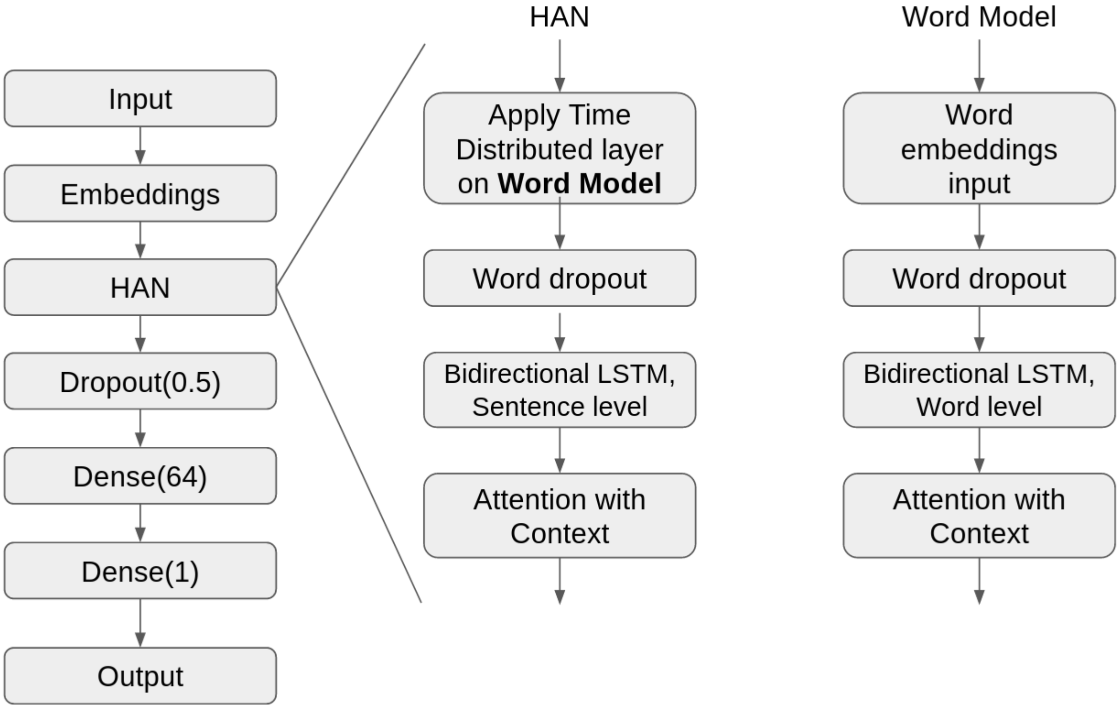


**Figure 5.6:** Figure showing the layer structure of HAN. The time distributed layer applies the word model on each sentence in a document.

## 5.4.2 Alpha SecureReqNet

The SRN implementation lacks an embedding layer and instead maps the document text data to their embedding values, reshapes the result, and feeds the embedding text into the neural network as input along with the max sentence length. The way embedded text is fed into the neural network is effectively the same as in the HAN model because the embedding layer in HAN is frozen, which means the weights cannot be changed during training. For illustrations and more details about this model, refer to the research paper on SRN(Palacio et al., 2019).

The first layer is a 7-gram convolutional layer, with a kernel size of seven words by the embedding dimensions. All the convolutional layers use a ReLU activation function. The resulting 32 vector feature maps are then fed into a max pooling layer, which is responsible for down sampling the patches of a feature map, taking the maximum value of each patch. The flatten layer takes the pooled tensor and flattens it into a one dimensional vector. The vector is reshaped to (32,1,1) followed by a 5-gram convolutional layer. Another max pooling and flatten layer resulting in a 64 feature column matrix. Three 3-gram convolutional layers followed by another max pooling and flatten layer to fully connect the vector.

The model has dense layers that serve to reduce the number of features and dropout layers to reduce overfitting. The final layer is a dense layer with an output dimension of 2 and the activation function is softmax. The reason softmax is used is that the prediction is chosen to be multiclass classification with two classes: the security and non-security class. Multiclass classification with two classes is often not needed as the same result can be achieved with a binary classification, the authors of the model may have good motivation to do so. This is in contrast to the previous models, where the prediction value was binary with one dimension. The output of SRN has been adjusted into a 1 dimension prediction at a later stage for consistent and more easily interpreted results. The typical output will be 1 or 0 instead of (1,0) or (0,1).

It is worth noting that the number of trainable features in the model in total is slightly below 100k with a training set of size slightly above 100k. When there is less training data than features in a model, the model may not able to learn the optimal hidden states.

### 5.4.3 Hierarchical Attention Virtual Adversarial Network

The HAN architecture is also expanded with a VAT-implementation. Hierarchical Attention Virtual Adversarial Network (HAVAN) still retained the HAN-layer structure, but with some extra SSL steps added to it. The embeddings are normalized using the formula in Equation 5.4. The calculation of $L_{v-adv}$ of Equation 5.6 is then added to the loss function as well as the option to perturb the embeddings of the model during a train step. In HAVAN both labeled and unlabeled data is used during training, making it an SSL-based approach. Labeled data is used for the standard loss function, while both unlabeled and labeled data are used for the VAT loss function.

Since the problem investigated in this thesis is a binary classification problem, Bernoulli distribution is used as the distributions in Equation 5.6. The model can be viewed in Figure 5.7.

## 5.5 Evaluation

Evaluation is intended to measure the performance of the finished, trained model. The usefulness of this model can be interpreted from the results below using the following methods. For benchmarking a model, F1 score is a valuable asset as it takes both precision and recall

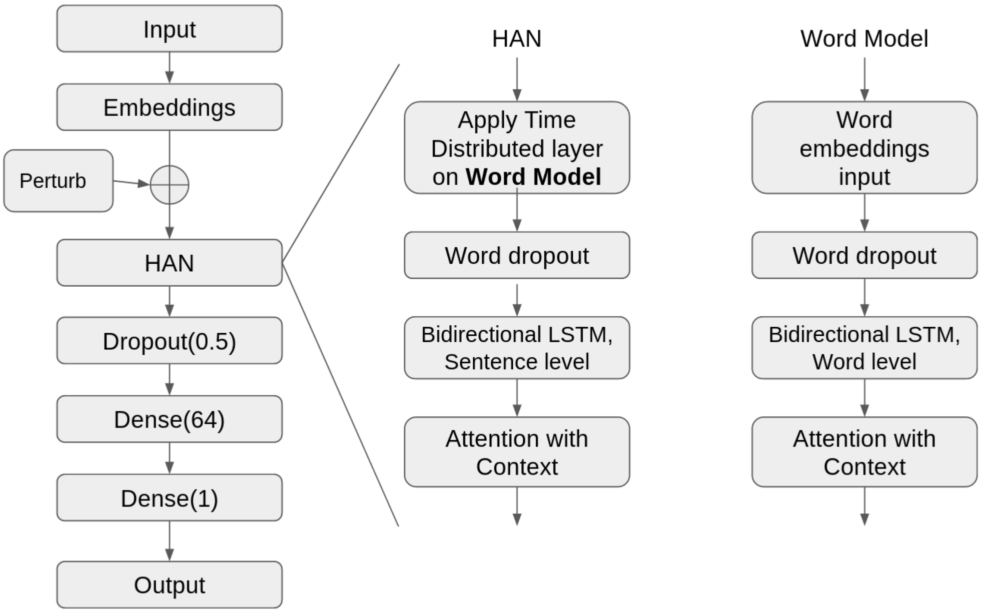


**Figure 5.7:** Figure showing the layer structure of HAVAN (HAN with VAT). Perturb is a perturbation that is added to the embeddings.

into its calculations. AUROC is used to plot the prediction results. In the evaluation, it is important to calculate the statistical significance of the results.

### 5.5.1 Metrics

The classifiers were evaluated on a test set of Github issues from the large user tagged, mixed source dataset, and separately the held-out gold standard data and the following metrics were recorded: precision, recall, and F1 scores for the positive and negative class. The relevant class for these metrics is primarily the positive class that encompasses security related text. Precision, recall, and F1 score are often used in scientific studies and will give more meaningful context to a predictor's performance than a simple accuracy score. There are several reasons to avoid accuracy, the most prominent being the way it can misrepresent performance on unbalanced test datasets. If only 1% of issues are security related, a model will achieve 99% accuracy by naively classifying none of the data as security related.

The mean and standard deviation of the evaluations per batch is intended to accurately represent the results. In the initial models, precision of security classifications was seen as one of the most important aspects, as a model with many false positives will waste a lot of human resources. A high precision classifier provides not only usefulness in industry applications, but also provides early insight into the difficulty of the task. While precision is essential, high recall is also important when satisfactory precision has been achieved. The final model comparisons will therefore use F1 score for security related classification.

### 5.5.2 Area Under the Receiver Operating Characteristics

The evaluation was also plotted as an Area Under the Receiver Operating Characteristics (AUROC). The curve is used to interpret how distinct the distributions for true positives and true negatives are. The overlap in the distributions describe difficulty in classifying the class correctly(Narkhede, 2018). AUROC has the benefit of comparing a random positive observation and seeing if it was classified as more positive than a random negative observation. This allows for a better representation of softer judgement that is useful for example if one wishes to use soft classification in the form of probabilities or scores relating to being positive or negative.

### 5.5.3 Statistical Properties

In order to quantify how well the classification results will represent performance on a larger dataset, statistical significance must be established. The size of the test data must be large enough to be able to make statements about the classification performance as a whole with at least 95% confidence.

### 5.5.4 Datasets

The evaluation consists of two datasets: Debricked Labeled Test Set and Github User Labeled Test Set. The sets are annotated under different policies, which will bring clarity as to how well the models detect more subtle signs of vulnerabilities. It also answers the question as to how well the model generalizes to other definitions of security. It is expected that the Debricked dataset is much more difficult and is not expected to produce good results. The model is trained on data similar to the Github User dataset and as such, it should perform much better on the Github User Labeled test set.

# Chapter 6

# Results

The evaluation results for each model will be presented in this section.

The comparisons of interest are:

- Utilization of training data - how much classification performance is gained from having a larger amount of training data?
- Weak Detection - do some models perform better on the less strict criteria defined in the annotation guidelines?
- Convergence rate - how quickly does the model learn the problem?
- Sensitivity to hyperparameter tuning.

There are two test sets used in the final evaluation of each model, User Labeled Test Set and Debricked Test Set. The results on these test sets can be seen in Table 6.1 and Table 6.2. Since we care more about the performance on the security related data, we prioritize the results on the security class above the macro average score.

As can be seen in Table 6.1, the best model when evaluating on community or user tagged Github issues is either HAN or the simple Logistic Regression. HAN achieves higher F1 score for security related content, while Logistic Regression was able to achieve slightly better precision. Note, there is low variance in performance when comparing the tested models on this test set.

When evaluating on the data annotated by the guidelines presented in this thesis, we observe that the HAVAN model is superior on this test set 6.2. The F1 score of HAVAN is only a few percent above HAN, but the precision is much higher. The highest recall was achieved by logistic regression but at the cost of much lower precision.

In Figure 6.2, the 95% confidence interval of the security-related results are evaluated. Observe that the Debricked Test Set evaluation is less accurate because there are much fewer observations in this test set. In future work, it would be interesting to expand this set in order to improve the correctness of the evaluation.

| User Labeled Test Set | Precision | Recall | F1 score | N |
|---|---|---|---|---|
| **LogisticRegression** | | | | |
| non-security related | 65% | 99% | 79% | 555 |
| security | **99%** | 42% | 59% | 514 |
| | | | | |
| macro average | 81% | 72% | 69% | 1069 |
| **SRN** | | | | |
| non-security related | 66% | 99% | 79% | 555 |
| security | 97% | 44% | 61% | 514 |
| | | | | |
| macro average | 81% | 71% | 70% | 1069 |
| **HAN** | | | | |
| non-security related | 68% | 99% | 80% | 555 |
| security | 97% | **49%** | **65%** | 514 |
| | | | | |
| macro average | **82%** | **74%** | **73%** | 1069 |
| **HAVAN (HAN w/ VAT)** | | | | |
| non-security related | 66% | 99% | 79% | 555 |
| security | 97% | 44% | 61% | 514 |
| | | | | |
| macro average | **82%** | 72% | 70% | 1069 |

**Table 6.1:** Table showing best results for each model on User Labeled Test Set. The User Labeled Test Set was annotated by users on GitHub. Bold entries shows the best result for security classification in each column.

The AUC scores can be seen in Table 6.3. Observe that the User Labeled Test Set achieves much better AUC scores and thus shows a much more distinct separation of the distributions of security and non-security data in comparison to Debricked Labeled Test Dataset. Note that the models are optimized for maximum validation accuracy, where the validation set contains User Labeled observations. The logistic regression approach achieved the best AUC score for both test set, closely followed by HAVAN. The Figure 6.1 shows ROC Curve of HAVAN on both test sets.

## 6.1 Statistical Significance

The confidence interval for the error on positive prediction was not too promising on the Debricked Test Set, as can be seen in Figure 6.2b. The confidence interval was quite large, which could be attributed to the small number of security related issues in the test set. Lack of human resources for annotating Github issues meant that this problem was not easily solved. In the future, we would like to expand this set to allow evaluation with more certain results. On the other hand, the confidence interval on the User Labeled Test Set was much smaller, meaning the evaluation is more precise.

| **Debricked Test Set** | **Precision** | **Recall** | **F1 score** | **N** |
|---|---|---|---|---|
| **LogisticRegression** | | | | |
| non-security related | 92% | 92% | 92% | 835 |
| security | 40% | **39%** | 40% | 112 |
| | | | | |
| macro average | 66% | 66% | 66% | 947 |
| **SRN** | | | | |
| non-security related | 91% | 89% | 90% | 835 |
| security | 30% | 36% | 33% | 112 |
| | | | | |
| macro average | 61% | 62% | 61% | 947 |
| **HAN** | | | | |
| non-security related | 91% | 97% | 94% | 835 |
| security | 57% | 33% | 42% | 112 |
| | | | | |
| macro average | 74% | 65% | 68% | 947 |
| **HAVAN (HAN w/ VAT)** | | | | |
| non-security related | 92% | 98% | 95% | 835 |
| security | **75%** | 35% | **48%** | 112 |
| | | | | |
| macro average | **83%** | **67%** | **71%** | 947 |

**Table 6.2:** Table showing best results for each model on Debricked test set. The Debricked test set was annotated by us, by using the Annotation Guidelines in the Appendix. Bold entries shows the best result for security classification in each column.

| | AUC |
|---|---|
| **LogisticRegression** | |
| User Labeled Test Set | **0.962** |
| Debricked Test Set | **0.729** |
| **SRN** | |
| User Labeled Test Set | 0.955 |
| Debricked Test Set | 0.634 |
| **HAN** | |
| User Labeled Test Set | 0.932 |
| Debricked Test Set | 0.666 |
| **HAVAN (HAN w/ VAT)** | |
| User Labeled Test Set | 0.939 |
| Debricked Test Set | 0.707 |

**Table 6.3:** Table showing the AUC score for each model and test set.

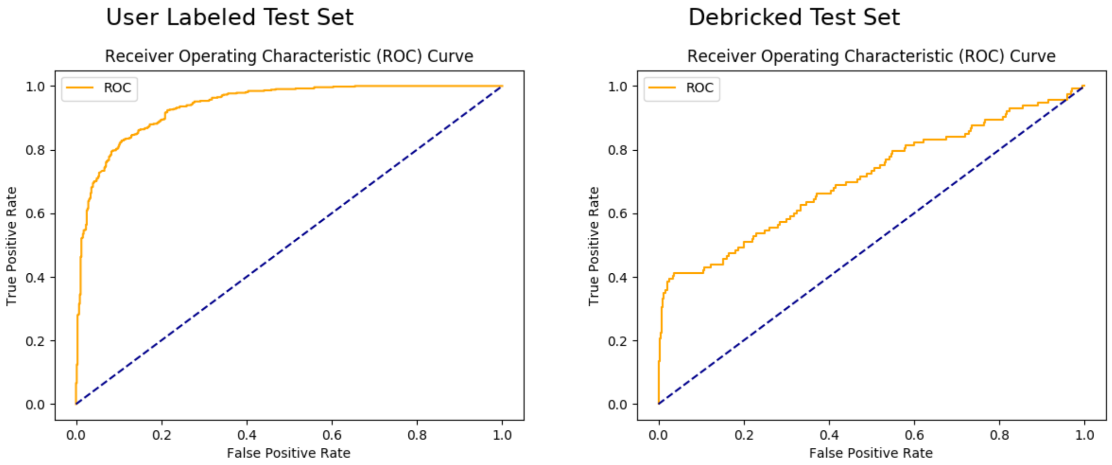


**Figure 6.1:** Graphs showing AUCROC of User Labeled Test Set, and AUCROC of Debricked Test Set on the HAVAN model.

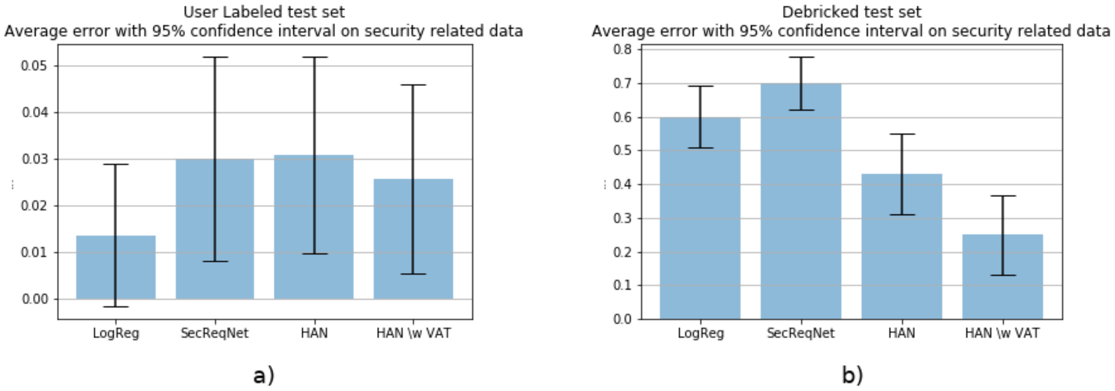


**Figure 6.2:** Figure showing the average error on security related data with its 95% confidence intervall. a) shows the User Labeled Set and b) shows the Debricked Test Set. The y-axes represents the average error, which is at most between 0 and 1, where 1 equals 100%. Note that the scales in the two graphs are different.

# Chapter 7
# Discussion

## 7.1 Data

In the exploratory data analysis stage, it was clear that the domains of NVD and Github had little overlap. This is considered during training and evaluation, as the models train primarily on NVD for security related text since security related Github issues are in short supply. Despite these issues, the HAN model was still able to achieve remarkable precision on security issues on Github. The mediocre recall can be attributed to the diversity in security related text and the many types of vulnerabilities that exist. It is possible that many types of vulnerabilities that appear in the test set have not appeared in the training set or that the text is phrased differently than CVE/CWE descriptions.

The results for Debricked Labeled test set and User Labeled test set vary greatly, with the models performing consistently worse on the Debricked Labeled set. This can be attributed in part due to the much more inclusive definition of security related, as seen in the Appendix. The models are hyperparameter optimized to maximize the validation accuracy, and the validation set contains a mix of data from a sample of the User Labeled set. The training data does not contain any data labeled according to the annotation guidelines constructed in this thesis. The User Labeled test set may be much easier to predict due to the security tagged Github data mostly being similar to the text in NVD data. Note that the annotations for Debricked Labeled test set do not consider discussion related to cybersecurity that is not indicative of risk to be security-related by tag. This includes suggestions or questions regarding security topics. It is possible that the models have trouble distinguishing security-related text that actually indicates risk as well as the harmless text. To better train the models to deal with this type of wrongful prediction, this type of data likely needs to be present in the training dataset.The User Labeled set has not been completely verified to be correctly annotated and relies on accurate tags from Github users. The VAT based HAN model had the best precision on the Debricked Labeled test set security category, which may be attributed to useful regularization making it more adaptable to problems similar to the training problem.

The models evaluated do not use the comments of each Github issue, only the description of the issue itself. This was done deliberately since the model should detect vulnerabilities at an early stage, before it gets tagged as security related. A better performance on test data could most likely be achieved by adding the comment texts to each Github data entry. It is possible that the clues to vulnerabilities are hidden in the comment section. This could lead to a lower recall as the model lacks context, but could also be one of the reasons why the precision on Debricked Test Set is lower. The Debricked Test set was annotated only based on the text in the title and in the description of the issue, while the model might have learned something that most security related issues have in common in the description even though it does not mention anything about security. Perhaps if the issues in Debricked Test set were annotated with full context of comments, we would have labeled some of them differently. Undiscovered vulnerabilities may exist in the safe class in training, validation, and test datasets. While the text itself may not seem security related to a human annotator, it is possible that the neural networks have found vulnerability patterns that may be difficult for humans to detect. Further analysis as to what issues are mislabeled could offer insight in regards to what is learned by the models.

## 7.2 Embeddings

The embeddings primarily used were created by Palacio et al. (2019). The intention was in part to represent the SRN results as favorably as possible, as well as saving time training our own embeddings. GloVe embeddings were temporarily tested as well with similar results. It is possible that training our own embeddings specific for the security text classification task can further improve the results presented in this thesis.

## 7.3 Evaluation

The model with the highest F1 score with proper hyperparameters ended up being the HAN model without VAT, as noted in the Result section, for the User Labeled Test Set. The best precision and F1 score on the Debricked Test Set is achieved by HAVAN. The claimed accuracy for SRN could not be achieved with the test data we used. Note that their open-source implementation was used with our data cleaning and preprocessing. Embedding solution had to be implemented by us as well. Hyperparameter tuning of the models was done to a near identical amount to make the comparisons as fair as possible. We reached out to the authors for their test data so we could benchmark against their claimed accuracy but were not able to acquire it. Therefore, the SRN model may perform better with different parameters or cleaning.

Several models achieve high precision but with mediocre recall. It may be possible to combine these models with an ensemble approach in order to achieve a much higher recall without sacrificing precision. The ensemble approach would be reliant on the models making different mistakes so that their combined recall would be higher.

It is possible to argue that some parties may prefer a high recall over precision if the model is relied on as a catch-all. Our models were not optimized for this situation.

### 7.3.1 Optimization and Training Philosophy

The classifiers are High Precision (HP) classifiers, prioritizing the precision on the security class. A high precision on security will result in few false positives that would otherwise waste precious time for cybersecurity personnel. This comes at the cost of lower recall meaning that many vulnerabilities will be left undetected. HP classifiers provide several benefits since they can be combined in an ensemble approach to increase recall on vulnerability detection assuming that the HP classifiers make different mistakes.

The results from training with varying hyperparameters gave widely different results for SRN, to a larger degree than HAN. It is possible that models that vary more in their results depending on hyperparameters have final results that are less representative of their potential prediction scores. With this aspect in mind, the SRN model may have more room for improvement than the HAN variant, which could provide context for its lower overall performance. A sensitive model requires more tuning until it reaches similar scores to an insensitive model and may result in more training time overall. Hyperparameter tuning is expensive, so insensitive models are preferable when possible.

## 7.4 Semi-Supervised Learning

Our implementation of VAT was not able to provide much better results than a model without it. Leveraging large unlabeled datasets is an endeavor that is worth continuing to pursue as most data is innately unlabeled and the amount of data available plays a large part in the learning potential of a given classification problem. Due to time constraints, the potential of VAT may not have been fully explored as different hyperparameters could be superior compared to the hyperparameters that best fit the base HAN model.

## 7.5 Mistakes and Bias

The temporal domain is not considered when splitting test and train datasets. This could give the models clairvoyant knowledge about future vulnerabilities, which could skew the results slightly. Therefore, the results may be more representative of classification accuracy of previously known vulnerability types. The test set was not engineered to contain every type of vulnerability, which may bias the results. A larger test set minimizes these concerns as more types of vulnerabilities will be present in a larger set.

The means of generating labeled data for training in sufficiently large quantities was underestimated and ultimately resulted in using data that was already tagged as being security related. Finding data that was related to computer security was time consuming. Few issues on Github relate to security and even fewer are tagged as security related by a user. The lack of balance in the distribution of security and non-security related Github issues meant that acquiring sufficient security issues with uniform sampling would take a very long time. From the uniformly sampled issues, only about 8% of the issues were vaguely security related in content. The security related part of the training set had to use CWE/CVE descriptions from vulnerability database entries.

## 7.6 Ethics

There are several ethical issues that must be mentioned in the context of machine learning. The data we use is taken from the internet without the creator having this use in mind. The data is gathered legally, but taking advantage of other people's work may be frowned up depending on the context. We believe our project has had the ethical aspects in mind and will serve the greater good.

In regards to the outsourcing of the data annotation process, we chose to opt out of this option because it is difficult to annotate without domain specific knowledge. We also thought it was important to experience this process for ourselves so we could have a better understanding for the work that goes into this step. We found that it was soul crushingly tedious and we did not enjoy it. This new perspective gives us a great appreciation for the work that goes into manual annotations.

# Chapter 8
# Final Thoughts

## 8.1 Conclusion

In this master thesis, we have expanded on the concept of using NLP for security text classification. While the problem of security text classification is undeniably a difficult task, there are still improvements that can be made and techniques to explore. We have proved the viability of the HAN architecture, designed for documents, in the domain. The concept of SSL in NLP in the domain of security has shown promising results, indicating that the vast unlabeled data can be leveraged in this task. VAT improved the performance of classification on the Debricked Test Set. The algorithms described can help reduce labeling cost and improve open-source security through automation.

The best performance on the User Labeled Test Set was achieved by the HAN model with 97% precision and 49% recall. In contrast, the best model for the industry test set (Debricked) was achieved by HAVAN at 75% precision and 35% recall. Considering that the performance on the user labeled test set was very similar for all the models and the performance varied substantially more for the industry test set, the HAVAN model was considered the best performing model in the end. The motivation behind this statement being that HAVAN performed very well on both test sets, but was the best model on the Debricked Test Set which was considered a more difficult set to classify.

## 8.2 Future Work

Though the results look promising, there are still a lot of improvements to investigate as future work. We did not have time to implement and evaluate all the techniques and concepts available, but suggest alternatives for additional research in this section.

The data cleaning step can be greatly improved by removing random noise such as tokens that occur too often or too seldom. Tokens that are underrepresented will not be something the model can learn from, for example those that occur only once. These tokens can be replaced by an Unknown token that will be present in a meaningful amount of documents. The same concept can be used to give value to numbers with a tag for perhaps years and version numbers.

Transfer learning on a language model utilizing ALBERT could prove promising. The more data available the more powerful this method should be.

The definitions of computer security risk that also counts potential exposures such as memory leaks and crashes is difficult to train for and the domains are somewhat different. Multiclass classification schemes may be more suited to the annotation guidelines that were created.

Hyperparameter tuning is an unending process, leaving room for further optimization.

An interesting future prospect is to combine vulnerability detection algorithms with a vulnerability classification model that can categorize the vulnerabilities by CWE descriptions. It is also possible to incorporate means of scoring these vulnerabilities with a Common Vulnerability Scoring System (CVSS) that aims to measure the severity of vulnerabilities(Jormakka, 2019).

### 8.2.1 Transfer Learning

Recent work(Devlin et al., 2018) in NLP shows that transfer learning is more than promising. As transfer learning revolutionized machine learning in other fields such as Computer Vision, it has in the past two years gained a lot of traction in NLP.

Just recently, ALBERT was released by Lan et al. (2019) and showed promising results that more parameters does not always translate to just better results. Even more recently T5 was released and showed that any natural language problem could be transformed into a sentence prediction problem(Raffel et al., 2019).

As future work, it would be interesting to see what fine-tuning T5 and ALBERT would do for our results.

### 8.2.2 Semi-supervised learning

With the enormous amounts of unlabeled data available online, the prospect of trying different SSL methods in the future is enticing.

The Semi-supervised learning method evaluated in this thesis was Virtual Adversarial Training. It mainly modified the loss function and was therefore possible to add to an existing model. Other SSL approaches studied were Semi-supervised Variational Auto-encoders (SSVAE)(Xu et al., 2016) and Discriminative Adversarial Networks (DAN)(dos Santos et al., 2017), but due to lack of time it was not implemented.

# Chapter 9
# Appendix

## 9.1 Annotation Guidelines

A policy was established in order to quicken the annotation process and ensure that similar annotations were made. All data in the gold standard was annotated by one of the authors of this thesis. The authors have moderate knowledge in the field of cybersecurity, a condition that must be met in order to adequately label data as relating to computer security. Some data was annotated by both parties and compared in the cases of mismatch to ensure the annotations were similar.

The task of annotating the issues was both hard and tedious. A lot of the issues were ambiguous and unclear, making it important to create a policy. An annotation guideline was worked on to establish a unified labeling method. It was updated regularly during the annotation phase whenever a new kind of case arose.

The following categories do not discriminate between questions, warnings, or other discussions about a certain topic. The text is annotated as the most severe category that accurately describes it. The priority goes from Vuln being highest to Safe being lowest.

**Vuln:** Presence of known exploits, user reported vulnerabilities.

**Risk:** Commonly exploited methods such as: unrestricted user input, memory leaks, unexpected/unintended r/w/e os/database access, overflows, user reported potential risk, segmentation fault, access violation.

**Caution:** Breaking changes, breaking dependencies, breaking compilation, breaking updates, installation issues, authentication problems, port or socket malfunctioning, firewall issues service unavailable, site down, failed tests, out of memory, crash due to instabilities, unexpected/unintended r/w/e os/database deny, broken links., unknown CPU usage (mostly high usage with no obvious reason for it), incorrect mathematical calculations (with potential side effects), runtime errors, unknown memory issues, configuration problems of server, error-flags concerning security, talks about computer security in some way.

**Unsure:** Unexpected behaviour, minor breaking changes (e.g new functionality that has

| Top Word Unigrams Descending Order | |
|---|---|
| **NVD (filtered)** | **Github** |
| allows | js |
| vulnerability | error |
| attackers | node |
| improper | version |
| arbitrary | file |
| cve | com |
| web | lib |
| site | using |
| execute | use |
| cross | src |
| service | function |
| memory | modules |
| buffer | https |
| scripting | line |
| cause | code |
| denial | app |
| information | new |
| sql | usr |
| input | issue |
| crafted | build |
| access | type |
| parameter | test |
| unspecified | http |
| allow | like |
| earlier | debug |
| neutralization | users |
| bounds | request |
| injection | github |
| attacker | object |

**Table 9.1:** Unigrams: single terms with no spaces.

not been used in production in previous version), lack of confidence in its safety, UI bugs, development mode only issues

**Safe:** Text does not cover topics concerning the categories above, such as issues asking for help with potential programming mistakes.

| Top Word Bigrams Descending Order | |
|---|---|
| **NVD (filtered)** | **Github** |
| remote attackers | node modules |
| allows remote | github com |
| cross site | youtube dl |
| execute arbitrary | usr lib |
| cve cve | py line |
| denial service | usr local |
| site scripting | https github |
| cause denial | steps reproduce |
| attackers execute | framework versions |
| improper neutralization | console log |
| arbitrary code | npm err |
| improper restriction | fff fff |
| bounds memory | com apple |
| memory buffer | dylib fff |
| attackers cause | file usr |
| operations bounds | expected behavior |
| restriction operations | lib python |
| web page | react native |
| input web | feature request |
| sql injection | library frameworks |
| page generation | linux gnu |
| neutralization input | module js |
| generation cross | index js |
| scripting xss | bug report |
| allow remote | src github |
| inject arbitrary | java org |
| arbitrary web | make sure |
| web script | usr bin |
| script html | latest version |

**Table 9.2:** Bigrams: pairs of terms separated by a space.

# 9.2 N-Grams

# 9.3 Sample Text Data

## 9.3.1 Before Cleaning

"3.6.3: Wrong number format after copy
past action <p>Run <code>SELECT TO_NUMBER('0.0000001969', '9999.9999999999') FROM dual</code><br> copy result to clipboard and past back to sql editor and you get <strong>1.969E-7</strong></p>"

### 9.3.2 After Cleaning

"wrong number format after copy past action run select to number from dual copy result to clipboard and past back to sql editor and you get e"

## 9.4 Most Common Words In Clusters

Clusters Cluster 0: git : [('site', 468), ('web', 421), ('page', 337), ('cross', 124), ('add', 95)] nvd : [('site', 16340), ('cross', 15690), ('web', 14419), ('scripting', 13506), ('remote', 12516)]

Cluster 1: git : [('like', 50097), ('use', 43732), ('add', 30520), ('way', 29108), ('using', 27821)] nvd : [('use', 906), ('number', 767), ('candidate', 755), ('reject', 754), ('consultids', 754)]

Cluster 2: git : [('function', 46234), ('return', 36355), ('code', 29743), ('var', 29735), ('error', 25113)] nvd : [('function', 337), ('pointer', 145), ('null', 144), ('dereference', 138), ('issue', 121)]

Cluster 3: git : [('version', 66493), ('expected', 58000), ('reproduce', 55980), ('steps', 52028), ('behavior', 40896)] nvd : [('issue', 137), ('os', 110), ('linux', 107), ('using', 107), ('information', 103)]

Cluster 4: git : [('text', 15564), ('like', 13093), ('using', 12237), ('html', 11632), ('css', 10889)] nvd : [('buffer', 8), ('issue', 8), ('width', 8), ('html', 7), ('using', 7)]

Cluster 5: git : [('js', 21429), ('node', 16274), ('file', 15831), ('webpack', 15559), ('use', 14492)] nvd : [('plugin', 18), ('wordpress', 12), ('module', 11), ('wp', 7), ('files', 6)]

Cluster 6: git : [('php', 22225), ('error', 21127), ('line', 19804), ('version', 19748), ('file', 16532)] nvd : [('php', 351), ('allows', 151), ('information', 146), ('file', 146), ('attackers', 140)]

Cluster 7: git : [('using', 16351), ('window', 13958), ('issue', 13596), ('like', 13333), ('version', 12148)] nvd : [('issue', 44), ('does', 41), ('linux', 39), ('user', 35), ('kernel', 33)]

Cluster 8: git : [('xcode', 13946), ('version', 13047), ('error', 12325), ('ios', 12197), ('build', 12076)] nvd : [('android', 127), ('versions', 80), ('id', 64), ('product', 61), ('privilege', 54)]

Cluster 9: git : [('error', 20556), ('src', 13511), ('version', 13480), ('main', 13039), ('run', 11743)] nvd : [('issue', 64), ('discovered', 63), ('kernel', 47), ('linux', 44), ('pointer', 34)]

Cluster 10: git : [('id', 15958), ('type', 15954), ('query', 12266), ('version', 11657), ('database', 11496)] nvd : [('id', 359), ('user', 351), ('users', 206), ('use', 184), ('password', 166)]

Cluster 11: git : [('com', 64751), ('https', 60539), ('github', 41509), ('http', 23746), ('issue', 14004)] nvd : [('com', 43), ('https', 41), ('http', 30), ('issue', 15), ('github', 14)]

Cluster 12: git : [('remote', 272), ('memory', 252), ('service', 151), ('allows', 150), ('allow', 148)] nvd : [('allows', 58536), ('remote', 50901), ('attackers', 48861), ('vulnerability', 36376), ('improper', 35862)]

Cluster 13: git : [('app', 8918), ('atom', 8468), ('version', 4396), ('js', 4082), ('file', 3947)] nvd : [('app', 80), ('user', 62), ('users', 58), ('local', 58), ('resources', 51)]

Cluster 14: git : [('file', 45117), ('error', 21490), ('version', 21175), ('files', 19919), ('using', 18173)] nvd : [('file', 980), ('users', 969), ('local', 968), ('allows', 575), ('files', 546)]

Cluster 15: git : [('react', 30831), ('component', 25160), ('using', 13295), ('render', 12735), ('use', 12213)] nvd : [('component', 27), ('issue', 7), ('versions', 7), ('vulnerable', 6), ('affected', 6)]

Cluster 16: git : [('node', 39042), ('js', 37974), ('error', 29139), ('modules', 27661), ('lib', 18784)] nvd : [('module', 74), ('node', 65), ('js', 52), ('information', 49), ('exposure', 44)]

Cluster 17: git : [('server', 25890), ('error', 24841), ('http', 18885), ('using', 17925), ('request', 17913)] nvd : [('server', 742), ('user', 446), ('information', 417), ('http', 355), ('access', 323)]

# Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.

Behl, D., Handa, S., and Arora, A. (2014). A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

dos Santos, C. N., Wadhawan, K., and Zhou, B. (2017). Learning loss functions for semi-supervised learning via discriminative adversarial networks.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Ferenc, R., Hegedüs, P., Gyimesi, P., Antal, G., Bán, D., and Gyimothy, T. (2019). Challenging machine learning algorithms in predicting vulnerable javascript functions. pages 8–14.

Github (2020). `https://github.com/`.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial networks.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples.

J. Pennington, R. Socher, C. D. M. (2014). Glove: Global vectors for word representation. `https://nlp.stanford.edu/projects/glove/`.

Jormakka, O. (2019). Approaches and challenges of automatic vulnerability classification using natural language processing and machine learning techniques.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kocmi, T. and Bojar, O. (2017). An exploration of word embedding initialization in deep-learning tasks.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations.

Lopez, M. M. and Kalita, J. (2017). Deep learning applied to nlp.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction.

Miyato, T., Dai, A. M., and Goodfellow, I. (2016). Adversarial training methods for semi-supervised text classification.

Miyato, T., ichi Maeda, S., Koyama, M., Nakae, K., and Ishii, S. (2015). Distributional smoothing with virtual adversarial training.

Narkhede, S. (2018). Understanding auc - roc curve. `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`.

Nguyen, M. (2018). Illustrated guide to lstm's and gru's: A step by step explanation. `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21`.

NIST (2020). National vulnerability database. `https://nvd.nist.gov/`.

Palacio, D. N., McCrystal, D., Moran, K., Bernal-Cárdenas, C., Poshyvanyk, D., and Shenefiel, C. (2019). Learning to identify security-related issues using convolutional neural networks.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

Rocca, J. (2019). Understanding variational autoencoders (vaes). `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

Ruder, S. (2016). An overview of gradient descent optimization algorithms.

Synopsys (2018). 2018 open source security and risk analysis synopsys cybersecurity research center. `https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf`.

Synopsys (2019). 2019 open source security and risk analysis synopsys cybersecurity research center. `https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-19.pdf`.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2014). Efficient object localization using convolutional networks.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Xu, W., Sun, H., Deng, C., and Tan, Y. (2016). Variational autoencoders for semi-supervised text classification.

Xuan, J., Jiang, H., Ren, Z., Yan, J., and Luo, Z. (2017). Automatic bug triage using semi-supervised text classification.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of cnn and rnn for natural language processing.

Zhang, Y. and Wallace, B. (2017). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Zou, D., Deng, Z., Li, Z., and Jin, H. (2018). *Automatically Identifying Security Bug Reports via Multitype Features Analysis*, pages 619–633.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-01-24

**EXAMENSARBETE** Semi-Supervised Text Classification
Automated Weak Vulnerability Detection
**STUDENTER** Anton Duppils, Magnus Tullberg
**HANDLEDARE** Marcus Klang (LTH), Emil Wåreus (Debricked)
**EXAMINATOR** Pierre Nugues (LTH)

# Textklassificering på delvis kategoriserad data: Automatisk svag sårbarhetsdetektering i text

POPULÄRVETENSKAPLIG SAMMANFATTNING **Anton Duppils, Magnus Tullberg**

Viktiga system digitaliseras. Beroendet på öppen källkod ökar. Övervakning av diskussion kring kod behövs för att snabbt detektera sårbarheter. Neurala nätverk kan automatisera detektionen genom att utnyttja diskussion kring projekt med öppen källkod.

Idag innehåller mer än 99% av kodbaser öppen källkod enligt en ny rapport från Synopsys. Utöver det så innehåller 40% av de undersökta kodbaserna öppen källkod med sårbarheter som är äldre än 10 år. Beroendet av öppen källkod gör det svårt att hålla koll på alla potentiella sårbarheter, samtidigt som en sårbarhet i koden kan ge förödande effekter. Att detektera och följa användarrapporterade sårbarheter är viktigt, då de flesta sårbarheterna som utnyttjas kommer från tidigare rapporterade svagheter.

Vi presenterar en ny maskininlärningsmetod för binär textklassificering; Är texten relaterad till datasäkerhet? Algoritmen kan användas för att detektera om ett foruminlägg är säkerhetsrelaterad. Algoritmen är ett neuralt nätverk med en uppnådd precision på 97% på säkerhetsrelaterade inlägg och lyckas hitta 47% av de totala sårbarheterna. Den inledande träningen är väldigt datorkraftskrävande, men när träningen är klar så är klassificering förhållandevis billig.

Det valda neurala nätverket är ett så kallat Hierarchical Attention Network (HAN), som är en arkitektur framtagen för textdokumentsklassificering. Vårt nätverk använder även Virtual Adversarial Training (VAT) för att utnyttja omärkt data under träningen. Det grundläggande konceptet med VAT är att lura nätverket att begå misstag under träningen för att få ett mer robust nätverk i slutändan. Modellen HAVAN - som använder HAN med VAT - jämfördes med tidigare kända starka klassificerare, såsom en logistic regression-klassificerare såväl som ett convolutional neural network. HAVAN visade sig ge bäst resultat givet liknande optimeringsmöjligheter.

Resultaten påvisar att maskininlärning kan nyttjas effektivt för att detektera sårbarheter genom textklassificering. Framtida studier kan svara på följande frågor: Vilka sorters sårbarheter är svåra att detektera och varför? Kan HAVAN-arkitekturen visa hög prestanda på standardiserade problem? Vad är optimeringsbegränsningarna för de neurala nätverken? Slutligen så ber vi läsaren att fundera över vart framtiden kan leda oss. Hur tror du att den ständiga framfarten inom maskininlärning kommer att förändra världen?