M.Sc. Thesis

“Toxicity Detection in User Generated Content”

Ioannis-Ion Koutsikakis

EY1606

Supervisors: Ion Androutsopoulos
Prodromos Malakasiotis
John Pavlopoulos

ATHENS, JUNE 2018
Abstract

In this thesis we experimented with deep learning models for toxicity detection in user generated content. By using novel architectures and extensive tuning, we surpassed the previous state of the art, namely an RNN with a self-attention mechanism, which was one of the first attempts to tackle this problem using deep learning. The architectures of this thesis include a multi-head attention RNN, a CNN, an RCNN, as well as a reimplementation of the attention RNN. Additionally we show that using ensemble models boosts the results even further. Finally we show a way to highlight per category “suspicious” words of a comment by exploiting the attention scores provided by a multi-attentional model that uses a separate attention mechanism for each category. The experiments were performed on a dataset containing comments from a Greek sports news portal that were accepted or rejected by the portal’s moderators, and on an English multi-label dataset that includes comments labeled for the type of toxicity, if any, they contain.

Περίληψη

Στην παρούσα διπλωματική εργασία έγινε πειραματισμός με μοντέλα βαθιάς μάθησης (deep learning) για την ανίχνευση της τοξικότητας σε περιεχόμενο παραγόμενο από χρήστες. Με τη χρήση πρωτοποριακών αρχιτεκτονικών και εκτεταμένης αναζήτησης των κατάλληλων υπερ-παραμέτρων (hyper-parameter tuning), ξεπεράστηκε η καλύτερη μέχρι τώρα τεχνολογία (state of the art), η οποία συνόδευται σε ένα Ανατροφοδοτούμενο Νευρωνικό Δίκτυο με μηχανισμό αυτο-προσοχής (self-attention RNN) που αποτελούσε μια από τις πρώτες προσπάθειες αντιμετώπισης του προβλήματος με χρήση βαθιάς μάθησης. Οι αρχιτεκτονικές αυτής της διπλωματικής εργασίας περιλαμβάνουν ένα RNN προσοχής με πολλαπλές κεφαλές (multi-head attention RNN), ένα Συνελικτικό Νευρωνικό Δίκτυο (CNN), ένα CNN το οποίο δέχεται ως είσοδο την έξοδο ενός RNN (RCNN), καθώς επίσης και μία επαναλειτουργία του RNN προσοχής. Επιπλέον, στην εργασία καταδεικνύεται ότι η ταυτόχρονη χρήση συλλογής μοντέλων (ensemble models) ενισχύει ακόμη περισσότερο τα αποτελέσματα. Τέλος, παρουσιάζεται ένας τρόπος με τον οποίο μπορούν να επισημανθούν ανά κατηγορία οι ‘ύποπτες’ λέξεις ενός σχολίου, με τη χρήση των βαθμολογιών προσοχής (attention scores) που παρέχονται από ένα μοντέλο το οποίο χρησιμοποιεί έναν ξεχωριστό μηχανισμό προσοχής ανά κατηγορία. Τα πειράματα διεξήχθησαν επί ενός συνόλου δεδομένων που περιέχει σχολία από μία ελληνική δικτυακή πύλη αθλητικών ειδήσεων, τα οποία είτε έγιναν αποδεκτά, είτε απορρίφθηκαν από τους διαχειριστές της πύλης, καθώς και επί ενός αγγλικού συνόλου διεθνούς που περιλαμβάνουν σχολία που έχουν επισημανθεί ανάλογα με την τοξικότητα που περιέχουν.
Acknowledgements

I would like to express my sincere thanks to my supervisor Ion Androutsopoulos for his constant support and encouragement and especially for giving me, once again, the opportunity to explore an area of great interest. I would also like to express my gratitude to Prodromos Malakasiotis for his valuable guidance throughout the development of this thesis as well as for many mind-clearing ping-pong sessions. In addition, I would like to thank John Pavlopoulos for the fruitful debates we had as well as George Brokos for his friendship and the interesting discussions we had regarding our works. Furthermore, I would like to thank AUEB’s NLP Group for many interesting meetings about cutting-edge technologies and StrainTek for their support during this thesis. Last but not least, I would like to thank my family for always being there for me.
Contents

Abstract i
Acknowledgements ii

1 Introduction 1
  1.1 Contribution ........................................... 1
  1.2 Outline ............................................ 2

2 Models 3
  2.1 Common Modules ....................................... 3
  2.1.1 Embedding Layer .................................. 3
  2.1.2 Projection Layer ................................. 5
  2.1.3 Output MLP ....................................... 5
  2.1.4 Dropout .......................................... 6
  2.1.5 Sequence Dropout ............................... 6
  2.2 Encoders ............................................ 6
  2.2.1 AttRNN-e ......................................... 6
  2.2.2 MHAttRNN-e ..................................... 8
  2.2.3 CNN-e ........................................... 9
  2.2.4 RCNN-e .......................................... 9

3 Experiments 13
  3.1 Datasets ........................................... 13
  3.1.1 MultiTox ......................................... 13
  3.1.2 Gazzetta ......................................... 15
  3.2 Word Embeddings .................................... 15
  3.3 Training ............................................ 17
  3.4 Baseline Model ..................................... 18
  3.5 Preliminary Experiments ............................ 18
  3.6 Hyper-Parameter Tuning ............................. 19
  3.7 Ensemble Models ................................... 19
  3.8 Results ............................................. 20
  3.9 Highlighting Suspicious Words .................... 24

4 Related Work 25

5 Conclusions 27
  5.1 Future Work ......................................... 27

Bibliography 29

Appendix A Model Architectures 32

Appendix B Examples of Highlighting Abusive Words 36
Chapter 1

Introduction

News portals and blogs often allow their readers to leave comments. It is essential for the good reputation of these websites that the comments they host are of good quality and do not include abusive content. For this reason the owners usually employ moderators that inspect each new comment and decide whether it should be allowed to be published or not. A problem that arises is that making this decision takes time, while comments arrive at great velocity. As a result comments are published with a delay upsetting, therefore, the users. On the other hand, many website owners can’t afford employing moderators. Consequently, there is an increased need for automated systems that are able to classify and reject abusive comments automatically or at least provide an indication to help the moderators decide faster. In addition, it is of scientific and industrial interest to be able to distinguish the type of toxicity that a comment contains. For example a portal may want to allow general profanity but reject comments containing personal attacks or identity hate.

The most basic systems use a list of bad words, which is created either by hand or by using simple machine learning methods. These systems discard comments that include at least one of the words contained in this list. The problem with this approach is that a comment can be abusive without containing any bad word. For example the sentence “I wish you cancer.” does not include any bad word but is clearly toxic. If the system was able to correctly classify this comment as toxic because of the word “cancer”, then it would also classify as toxic the sentence “Cancer is one of the leading causes of death.” which would be a mistake. For this reason we need to resort to more sophisticated systems that use contextual information to draw conclusions about the comments.

1.1 Contribution

Recently, deep learning based models (Goldberg, 2017; Goodfellow et al., 2016) where proposed to tackle the tasks discussed above. Pavlopoulos et al. (2017a) proposed a word based Recurrent Neural Network (RNN) (Cho et al., 2014; Hochreiter and Schmidhuber, 1997) with an attention mechanism (Yang et al., 2016; Mnih et al., 2014; Bahdanau et al., 2016; Xu et al., 2015; Luong et al., 2015), which surpassed the previous state of the art for abusive content detection on a dataset containing comments posted on the Wikipedia Talk pages, namely a Multi Layer Perceptron (MLP) operating on top of character n-grams introduced by Wulczyn et al. (2017). The attention RNN model was reimplemented in this thesis alongside three other deep learning models; an attention RNN that uses more than one attention scores for each word, a word based Convolutional Neural Network (CNN) proposed by Kim (2014), and an RCNN where a CNN operates on the output of an RNN. Moreover, a projection layer, which moves the input to a more appropriate space for the task at hand, was introduced. These models were tuned and compared to the previous architecture of the attention
RNN on the GAZZETTA dataset (Pavlopoulos et al., 2017a; Pavlopoulos et al., 2017b), which contains comments from the Greek sports news portal Gazzetta alongside the decisions of a moderator,1 and on the MULTITOX dataset which corresponds to a multi-label task of detecting toxic comments alongside the type of toxicity that they contain. Furthermore, using ensemble models for MULTITOX had a positive impact on the results. Additionally, it is shown that by using a multi-attentional model, that uses a separate attention mechanism for each category, we can highlight suspicious words per category, something that helps moderators to reach a conclusion even faster. The models described in this thesis were part of the ensemble with which StrainTek2 participated in the Kaggle’s Toxic Comment Classification Challenge.3 We achieved the 35th position out of 4551 teams. The code accompanying this thesis is available on GitHub.4

1.2 Outline

The rest of the thesis is organized as follows:

• Chapter 2 describes the architectures of the models.
• Chapter 3 presents the training methodology alongside the results of the experiments.
• Chapter 4 discusses related work.
• Chapter 5 draws the conclusions and proposes ideas for future work.

---

1In this thesis an extended version of this dataset was used; see Section 3.1.2 for more details.
2See https://straintek.wediacloud.net/
3See https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
4See https://github.com/jkoutsikakis/toxicity-detection-in-user-generated-content
Chapter 2
Models

The models take as input a user comment $C$ and output the probabilities of $C$ to belong to each category of the corresponding task. All the models convert the words of the comment to their corresponding embeddings, project those embeddings to another space through a projection layer, and feed those projections to an encoder in order to construct a dense representation for the comment. This dense representation is fed to an MLP that outputs the final probabilities. All the models are of similar structure, but they use different encoders. An abstract model is pictured in Figure 2.1. First, the modules that are common in all models are described (Section 2.1) and then the models themselves alongside their respective encoders (Section 2.2).

2.1 Common Modules

2.1.1 Embedding Layer

The exact input of the models is a sequence of $T$ vectors $<w_1, w_2, \ldots, w_T>$ that correspond to the words of a comment. Every $w_i$ is a one-hot vector that encodes a specific word from the vocabulary. The Embedding Layer is a matrix $E \in \mathbb{R}^{d \times \|V\|}$ where $V$ is the vocabulary and $d$ is the size of the embeddings, that when multiplied with a one-hot vector $w_i$ results to the corresponding word embedding $\epsilon_i$ as shown in Equation 2.1.

$$\epsilon_i = Ew_i,$$  \hspace{1cm} (2.1)

where $i \in \{1, \ldots, T\}$.

The matrix $E$ can be initialized randomly and altered during training but the most common practice is to initialize it with weights trained on an, often unsupervised, auxiliary task (Mikolov et al., 2013; Pennington et al., 2014; Bojanowski et al., 2017) which captures generally useful syntactic and semantic information. In the second case the embeddings can be either kept fixed or altered during training in the subsequent task. A problem that arises when we let the embeddings to be altered is that embeddings that are close with each other initially may end up far apart after training on the main task. This usually happens in small datasets, because some words that are contained in the unlabeled dataset of the auxiliary task are missing from the training dataset of the main task. For example, if the words “TV” and “Television” are contained in the unlabeled corpus, it is expected that the resulting word embeddings would be close to each other. If only the word “Television” is present in the main dataset then it will move away from the embedding of the word “TV” which stays the same. An additional benefit of keeping the embeddings fixed is that the models can be trained much faster since they use fewer parameters.
Figure 2.1: Illustration of the general model architecture. The words are first converted to embeddings and then passed through a projection layer before being fed in the encoder. Lastly, the encoded representation is given to an output MLP for the final decision.
2.1.2 Projection Layer

The next step is to project the embeddings to another space of \( m \) dimensions. Using a projection layer, we allow the embeddings to be moved to a more convenient space for the task without requiring the embeddings to be altered. The projection is a simple linear transformation plus a bias with a non-linearity \( \phi \) on top, as shown in Equation 2.2, where \( W_p \in \mathbb{R}^{m \times d} \), \( e_i \in \mathbb{R}^d \), \( b_p \), \( p_i \) \( \in \mathbb{R}^m \), \( d \) is the embedding size, and \( m \) is the projected embedding size. The possible non-linear functions that were considered are shown in Equations 2.3–2.5, where \( \tanh \) is the hyperbolic tangent, \( \text{relu} \) is the Rectified Linear Unit (Nair and Hinton, 2010), \( \text{prelu} \) is the Parametric Rectified Unit (He et al., 2015), \( \alpha \in \mathbb{R} \) is a trainable parameter, and \( \odot \) denotes element-wise multiplication.

\[
p_i = \phi(W_p e_i + b_p) \quad (2.2)
\]
\[
tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.3)
\]
\[
\text{relu}(x) = \max(x, 0) \quad (2.4)
\]
\[
\text{prelu}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha \odot x, & \text{otherwise} \end{cases} \quad (2.5)
\]

where \( i \in \{1, \ldots, T\} \).

2.1.3 Output MLP

The output MLP is a sequence of applications of \( l \) linear transformations plus a bias alongside \( l \) non-linear functions (activations). The first \( l - 1 \) applications are called hidden layers while the last one is called the output layer. The responsibility of this MLP is to predict the probability of the comment to belong to each of \( c \) classes given its vector representation produced by the encoder. Specifically, given the encoding \( z \in \mathbb{R}^k \) of size \( k \), the output MLP produces a vector \( \tilde{y} \in [0, 1]^c \) containing the \( c \) probabilities, as shown in Equations 2.6–2.12

\[
v_1 = \phi(W_1 z + b_1) \quad (2.6)
\]
\[
\vdots
\]
\[
v_j = \phi(W_j v_{j-1} + b_j) \quad (2.7)
\]
\[
\vdots
\]
\[
v_{l-1} = \phi(W_{l-1} v_{l-2} + b_{l-1}) \quad (2.9)
\]
\[
\tilde{y} = \sigma(W_l v_{l-1} + b_l) \quad (2.10)
\]
\[
\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.12)
\]

The last non-linearity is a sigmoid in order for the output to be a vector containing values between zero and one. The weights \( W_1 \in \mathbb{R}^{r \times k} \), \( W_j \in \mathbb{R}^{r \times r} \), \( W_l \in \mathbb{R}^{c \times r} \), \( b_1, b_j \in \mathbb{R}^r \), \( b_l \in \mathbb{R}^c \), are learned during training, where \( l \) is the depth, \( r \) is the size of the hidden layers, \( k \) is the size of the encoding, and \( \phi \) is a non-linear function. The possible choices for the non-linear function considered for the hidden layers are the \text{relu} and \text{prelu} (Equations 2.4 and 2.5).
2.1.4 Dropout

Srivastava et al. (2014) noticed that one of the reasons the Deep Learning models tend to overfit the training set is the so called co-adaptation of neural activations. In other words, the models tend to use specific combinations of activations in order to memorize the training set. A way they found to overcome this problem is to discard randomly in each iteration of the training algorithm some neural activations (zero the outputs of the corresponding neurons) with some predefined probability \( p \). This method is called Dropout and is used only during training after each hidden layer of the MLPs. During inference, the activations of the hidden layers are multiplied by \( p \) to ensure that the neurons of subsequent layers are fed with inputs of the same expected sum (same expected pre-activation).

2.1.5 Sequence Dropout

Sequence Dropout is similar to the regular Dropout except that it is used for sequences of vectors, like those given as input to RNNs. The difference lies in the fact that the same dimensions are dropped from all the vectors of the sequence. This avoids adding too much noise to the data since the same dimensions are kept in all the vectors in the same forward pass.

2.2 Encoders

2.2.1 AttRNN-e

The AttRNN-e encoder first converts the projected embeddings to context-aware ones using a Recurrent Neural Network with Gated Recurrent Units (RNN-GRU) and then creates a dense representation using a self-attention layer.

The RNN-GRU is a flavor of RNN invented by Cho et al. (2014), where the simple recurrent units are replaced with GRU ones. These units have been shown to be robust against the vanishing gradient problem; i.e., gradients that flow in very deep networks tend to approach zero (vanish) as we move away from the final layers during back-propagation (Pascanu et al., 2013b). The RNN-GRU was inspired by the RNN-LSTM (Hochreiter and Schmidhuber, 1997) but is simpler and faster to train, often without any noticeable difference in performance.

The RNN-GRU takes as input a sequence of \( T \) projected embeddings \(< p_1, p_2, \ldots, p_T >\) and outputs a sequence of \( T \) hidden states \(< h_1, h_2, \ldots, h_T >\) where \( h_i \in \mathbb{R}^k \), as shown in Equations 2.13–2.16. The vector \( z_i \in \mathbb{R}^k \) is called the update gate and controls how much new information should be added to the new hidden state and the \( r_i \in \mathbb{R}^k \) is called the reset gate and controls how much information should be discarded from the previous hidden state \( h_{i-1} \). The symbol \( \odot \) represents element-wise multiplication, and \( W_z, W_r, W_h \in \mathbb{R}^{k \times m}, U_z, U_r, U_h \in \mathbb{R}^{k \times k}, \) and \( b_z, b_r, b_h \in \mathbb{R}^k \) are trainable parameters.

\[
\begin{align*}
z_i &= \sigma( W_z p_i + U_z h_{i-1} + b_z ) \\
r_i &= \sigma( W_r p_i + U_r h_{i-1} + b_r ) \\
h^{\tilde{}}_i &= \tanh( W_h p_i + U_h ( r_i \odot h_{i-1} ) + b_h ) \\
h_i &= ( 1 - z_i ) \odot h_{i-1} + z_i \odot h^{\tilde{}}_i 
\end{align*}
\]

where \( i \in \{1, \ldots, T\} \).
For brevity, we denote the operation of RNN-GRU as:

\[
    h_1, h_2, \ldots, h_T = \text{rnn}_{\text{gru}}(p_1, p_2, \ldots, p_T)
\]  

(2.17)

There are two augmentations of the RNN that are used extensively in literature and often boost the performance of the model.

The first one is to make it bidirectional (Schuster and Paliwal, 1997). In this case we have two RNNs; the first accepts recurrently the input from left to right while the second from right to left. The hidden states of each direction are concatenated to form the final hidden states (Equations 2.18–2.20). The symbol ; denotes concatenation. As such, the hidden state of each time-step is affected from the whole sequence in contrast to the unidirectional RNN where it is affected only from the previous ones.

\[
    \vec{h}_1, \vec{h}_2, \ldots, \vec{h}_T = \text{rnn}_{\text{gru}}(p_1, p_2, \ldots, p_T)
\]  

(2.18)

\[
    \underline{h}_1, \underline{h}_2, \ldots, \underline{h}_T = \text{rnn}_{\text{gru}}(p_1, p_2, \ldots, p_T)
\]  

(2.19)

\[
    h_i = [\vec{h}_i; \underline{h}_i]
\]  

(2.20)

where \( i \in \{1, \ldots, T\} \).

The second augmentation is stacking (Pascanu et al., 2013a). In this case we have \( s \) RNNs that are stacked on top of each other so that the hidden states of the first RNN are fed as input to the second one and so on (Equations 2.21–2.23).

\[
    h^1_1, h^1_2, \ldots, h^1_T = \text{rnn}^1_{\text{gru}}(p_1, p_2, \ldots, p_T)
\]  

(2.21)

\[
    h^j_1, h^j_2, \ldots, h^j_T = \text{rnn}^j_{\text{gru}}(h^{j-1}_1, h^{j-1}_2, \ldots, h^{j-1}_T)
\]  

(2.22)

\[
    h^s_1, h^s_2, \ldots, h^s_T = \text{rnn}^s_{\text{gru}}(h^{s-1}_1, h^{s-1}_2, \ldots, h^{s-1}_T)
\]  

(2.23)

where \( j \in \{2, \ldots, s-1\} \).

A self-attention layer computes a weighted sum of its input. There are many ways to design a self-attention mechanism (Yang et al., 2016; Mnih et al., 2014; Bahdanau et al., 2016; Xu et al., 2015; Luong et al., 2015). Following Pavlopoulos et al. (2017a), we use a self-attention mechanism that first produces the summation weights \( \tilde{a} \in \mathbb{R}^T \), by applying an MLP on each input, which in this case are the hidden states \( <h_1, h_2, \ldots, h_T> \) of the RNN-GRU, where \( h_i \in \mathbb{R}^k \). The MLP can have as many hidden layers as we want but it must produce a single unnormalized score (Equation 2.24). Next, the softmax function is applied in order for the weights to sum up to one (Equations 2.25 and 2.27). Lastly, a weighted sum is performed, using the normalized scores as weights in order to create the final representation. The resulting representation \( z \in \mathbb{R}^k \) is an aggregation of information about the comment that is relevant to the task at hand. The ATTRNN-e encoder is depicted in Figure 2.2 and the model that uses it is named ATTRNN-m.
Figure 2.2: Illustration of ATTRNN-e, using a single unidirectional RNN-GRU for simplicity. First, the projected embeddings are fed to an RNN-GRU. Next, an MLP is used in order to create a single attention score for each context aware embedding created by the RNN-GRU. All the scores produced are fed to a softmax function in order to sum up to one. Finally, the weighted sum of the context aware embeddings is computed, using the attention scores as weights, in order to create the dense representation of the comment.

\[
\tilde{a}_i = \text{mlp}_{\text{att}}(h_i) \tag{2.24}
\]

\[
a_i = \text{softmax}(\tilde{a}_1, ..., \tilde{a}_T) \tag{2.25}
\]

\[
z = a_1 h_1 + ... + a_T h_T \tag{2.26}
\]

\[
\text{softmax}(x_i | \tilde{x}_1, ..., \tilde{x}_T) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.27}
\]

where \(i \in \{1, \ldots, T\}\).

2.2.2 MHAttrRNN-e

The Multi Head Attention RNN encoder (MHATTRNN-e) is very similar to the ATTRNN-e encoder, but it uses more than one weights for each input vector (Vaswani et al., 2017). Specifically, the attention MLP now outputs \(f\) attention weights for each input vector and during the weighted summation each weight is multiplied with a portion of this vector, as shown in Equations 2.28–2.32, where \(\tilde{a}_i \in R^f\) is now a vector containing the unnormalized scores of the \(f\) heads for the input vector \(h_i\), \(a_i\) is the corresponding vector with the \(f\) normalized scores for \(h_i\), and \(z \in R^k\) is the final representation of...
the comment. For example if we have \( h_i \) vectors of \( k = 128 \) dimensions and \( f = 4 \) attention heads then the first attention head considers dimensions 1–32 of the \( h_i \) vectors, the second head dimensions 33–64, and so on. In this way we are giving the model the ability to create better representations since we allow it to give different attention scores in different parts of the input vectors. The MHATTRNN-e encoder is depicted in Figure 2.3 and the model that uses it is named MHATTRNN-m.

\[
\hat{a}_i = \text{mlp}_{att}(h_i) 
\]

(2.28)

\[
a_{ij} = \text{softmax}(\hat{a}_{ij}|\hat{a}_{1,j}, ..., \hat{a}_{T,j})
\]

(2.29)

\[
\text{head}^j = \sum_i a_{ij}h_i[(j-1)r + 1 : jr]
\]

(2.30)

\[
r = k \div f
\]

(2.31)

\[
z^\top = [\text{head}^1; \ldots; \text{head}^f]
\]

(2.32)

where \( i \in \{1, \ldots, T\} \), \( j \in \{1, \ldots, f\} \)

### 2.2.3 CNN-e

The CNN-e encoder (Kim, 2014) creates the dense representation of each comment by applying a series of convolutions to the projected embeddings. A convolution is defined as a filter \( w \in \mathbb{R}^{nm} \), a bias \( b \in \mathbb{R} \), and a non-linear function which are applied to a window of \( n \) projected embeddings of size \( m \). The window is slid across the input producing one number for each application of the convolution to the window. Then a max-over-time pooling operation is performed which outputs the largest of the produced numbers. Using \( k \) convolutions (possibly of different window size \( n \)) and performing max-over-time pooling for each one of them, results to a vector \( z \in \mathbb{R}^k \) that can be used as the final representation of the comment, as shown in Equations 2.33–2.35, where \( M \in \mathbb{R}^{T \times m} \) is a matrix containing the projected embeddings. Intuitively each convolution searches for a specific latent n-gram. If this latent n-gram is present somewhere in the comment then the corresponding dimension of the final representation will contain a large number. The CNN-e encoder is depicted in Figure 2.4 and the model that uses it is named CNN-m.

\[
\tilde{o}^j = \text{relu}(w_i^\top M_{(j:j+n-1)} + b)
\]

(2.33)

\[
o^j = \text{max}_{i}(\tilde{o}^j)
\]

(2.34)

\[
z^\top = [o^1, o^2, \ldots, o^k]
\]

(2.35)

where \( i \in \{1, \ldots, k\} \), \( j \in \{1, \ldots, T - n + 1\} \)

### 2.2.4 RCNN-e

The RCNN-e encoder is similar to CNN-e except that we feed the CNN with the outputs of an RNN-GRU instead of the projected embeddings directly, as shown in Equations 2.36 and 2.37. It is hoped that applying the CNN on top of context aware representations of words produced by an RNN-GRU will result in more robust filters since these representations are affected by the whole sequence. The model that uses this
Chapter 2. Models

Figure 2.3: Illustration of mhattRNN-e, using a single unidirectional RNN-GRU for simplicity. First, the projected embeddings are fed to an RNN-GRU. Next, an MLP is used in order to create multiple attention scores for each context aware embedding created by the RNN-GRU. All the scores that correspond to the same part of the context aware embeddings are fed to the same softmax function in order to sum up to one. Finally, the weighted sums of the corresponding parts of the context aware embeddings are computed, using the corresponding attention scores as weights, and the weighted sums are concatenated to create the dense representation of the comment.
Figure 2.4: Illustration of CNN-e. In this example we have three filters, where two of them (red and green) are of size 2 while the third (orange) is of size 3. We first organize the projected embeddings as the rows of a matrix. Each filter is slid and applied vertically, with each application resulting to a single number. As such, each filter creates a vector containing the results of all its applications. Finally, using max-over-time pooling we choose the maximum value of each vector. The vector containing these maximum values is considered the dense representation of the comment.
encoder is named RCNN-m.

\[
\begin{align*}
  h_1, h_2, \ldots, h_T &= \text{rnn}_{\text{gru}}(p_1, p_2, \ldots, p_T) \\
  z &= \text{cnn}(h_1, h_2, \ldots, h_T)
\end{align*}
\]
Chapter 3

Experiments

3.1 Datasets

In this section we discuss the two datasets used in this thesis. The first one is called MULTITOX and corresponds to a multi-label classification task in which we want to identify what types of toxicity a comment contains. The second one is called GAZZETTA and corresponds to a binary classification task in which we want to decide whether a comment should be allowed to be posted on Gazzetta, a Greek sports news portal.\(^1\)

3.1.1 MultiTox

The MULTITOX dataset includes comments from the Wikipedia Talk pages alongside their gold labels for each category (Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate). The categories are not mutually exclusive; i.e. a comment may belong to one or more categories. For example, an insult may be expressed in an obscene way, or comments expressing identity hate may also contain threats against the hated identity.

This dataset was assembled by the Conversation AI team,\(^2\) a research initiative founded by Jigsaw,\(^3\) and was used originally as the training set for Kaggle’s Toxic Comment Classification Challenge.\(^4\) Since the gold labels of the competition’s test set are not publicly available at the time of writing, only the training set was used in this thesis.

The original training set was shuffled and split into four parts; MULTITOX-TRAIN (99,571 examples) used for training the models, MULTITOX-VAL (20,000 examples) used for early-stopping, MULTITOX-DEV (20,000 examples) used for model selection, and MULTITOX-TEST (20,000 examples) used for the final evaluation.

Figure 3.1 shows the percentage of the dataset that was allocated for each set, Figure 3.2 shows the distribution of each category for each set, and Table 3.1 shows the average comment length, the number of types (i.e. unique tokens), and the number of tokens for each split as well as for the whole dataset.

Another subset of the Wikipedia Talk pages, which contains approximately 54.9 million unlabeled user comments from the same domain as the labeled corpus, was used for pre-training the word embeddings.\(^5\)

---

\(^1\)See \url{http://www.gazzetta.gr}

\(^2\)See \url{https://jigsaw.google.com/projects/#conversation-ai}

\(^3\)See \url{https://jigsaw.google.com/}

\(^4\)See \url{https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge}

\(^5\)See \url{https://figshare.com/articles/Wikipedia_Talk_Corpus/4264973}
Chapter 3. Experiments

Figure 3.1: MULTITOX Dataset Split.

Figure 3.2: MULTITOX Labels’ Distributions.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Dev</th>
<th>Val</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg length</td>
<td>77</td>
<td>79</td>
<td>76</td>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td># types</td>
<td>127,746</td>
<td>51,370</td>
<td>50,503</td>
<td>50,538</td>
<td>168,381</td>
</tr>
<tr>
<td># tokens</td>
<td>7,644,806</td>
<td>1,579,768</td>
<td>1,526,867</td>
<td>1,524,904</td>
<td>12,276,345</td>
</tr>
</tbody>
</table>

Table 3.1: Additional Information for MULTITOX.
### Table 3.2: Additional Statistics for GAZZETTA.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Dev</th>
<th>Val</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg length</td>
<td>44</td>
<td>48</td>
<td>44</td>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td># types</td>
<td>462,296</td>
<td>184,915</td>
<td>177,790</td>
<td>181,356</td>
<td>585,266</td>
</tr>
<tr>
<td># tokens</td>
<td>64,269,227</td>
<td>13,118,946</td>
<td>12,009,301</td>
<td>12,371,937</td>
<td>101,769,411</td>
</tr>
</tbody>
</table>

#### 3.1.2 Gazzetta

The GAZZETTA corpus includes user comments from the Greek sports news portal Gazzetta. The portal employs moderators who decide whether user submitted comments should be posted online or not. The moderators reject comments that are toxic, contain hate speech, bullying, irony and in general comments that could spark a cyber quarrel. Each example in the dataset includes a user’s comment alongside the decision of the moderator (Rejected, Accepted).

The dataset includes comments from the time period between 01-01-2015 and 19-10-2017. Because of the nature of the data (comments concerning current affairs), it is important that the trained model is time-robust; i.e. able to correctly classify comments concerning future events. As such, in order to come to the right conclusions with regard to the performance of the model, it is imperative that the dataset we use for the evaluation is subsequent to the one the model is trained on.

Hence, the training set contains comments from 01-01-2015 to 02-11-2016 (1,459,367 examples), the validation set contains comments from 02-11-2016 to 20-03-2017 (271,490 examples), the development set contains comments from 20-03-2017 to 30-06-2017 (271,490 examples) and the testing set contains comments from 30-06-2017 to 19-10-2017 (271,492 examples).

This dataset is an extended version of the dataset introduced by Pavlopoulos et al. (2017a). This new version uses all the comments of the previous one as training examples and approximately 814K additional comments for evaluation. Currently, this new version is not publicly available because we need permission from the Gazzetta News Portal which owns the data. It might be released in the future.

Figure 3.3 shows the percentage of the dataset that was allocated for each set, Figure 3.4 shows the percentage of comments that got rejected for each set and Table 3.2 shows the average comment length, the number of types, and the number of tokens for each split as well as for the whole dataset.

Another subset of the Gazzetta corpus was used for pre-training the word embeddings, which contains approximately 5.2 Million unlabeled user comments. This is the same unlabeled corpus that was used by Pavlopoulos et al. (2017a) for the same reason.

#### 3.2 Word Embeddings

In the case of GAZZETTA, the Embedding Layer was pre-trained using WORD2VEC (Mikolov et al., 2013) in the Greek unlabeled dataset and remains fixed during training on the subsequent task. The WORD2VEC algorithm has the effect that embeddings of semantically and syntactically similar words end up close together in space. Also they display some interesting linear properties, e.g. the embedding of the word “king” is approximately equal to the vector resulting by subtracting the embedding of
Chapter 3. Experiments

Figure 3.3: GAZZETTA Dataset Split.

Figure 3.4: GAZZETTA Rejection Label Distribution.
“woman” from that of “man” and adding the one of “queen” (Equation 3.1).

\[ \epsilon(\text{KING}) \approx \epsilon(\text{MAN}) - \epsilon(\text{WOMAN}) + \epsilon(\text{QUEEN}) \]  

(3.1)

In case of MULTITOX, the matrix \( E \) was pre-trained using FASTTEXT (Bojanowski et al., 2017) in the English unlabeled dataset and remains fixed during training on the subsequent task as well. The FASTTEXT algorithm is similar to WORD2VEC but it additionally takes into account all possible character \( n \)-grams (\( n \) consecutive characters) of each word when generating its embeddings instead of treating each word as an atomic unit. As such, it is robust to grammatical errors and handles well out of vocabulary words that are close both grammatically and semantically to known words.

Preliminary experiments showed that FASTTEXT embeddings performed better than WORD2VEC in MULTITOX in contrast to GAZZETTA where WORD2VEC performed best. Since FASTTEXT considers all possible character \( n \)-grams, it uses a lot more parameters than WORD2VEC. As such it needs to be trained on a larger dataset (like the English one) in order to perform well. In both cases, the CBOW variant of the respective algorithm was used with a context window of size 5, and embeddings of 300 dimensions were produced. We used the gensim\(^6\) and fasttext\(^7\) Python libraries for their WORD2VEC and FASTTEXT implementations respectively.

### 3.3 Training

The initial weights, apart from those of the Embedding Layer, are sampled using the GLOROT-UNIFORM method as shown in Equation 3.2, where \( W \in \mathbb{R}^{fan_{\text{out}} \times fan_{\text{in}}} \) is a matrix containing the weights of a linear transformation and \( \text{unif} \) is the uniform distribution. Glorot and Bengio (2010) showed that using this initialization scheme has the effect that at least in the beginning of the training phase, the variance of the output of a linear transformation is very close to the variance of its input. This is important because it keeps the signal from exploding to a high value or vanishing to zero, something that would lead to instability of the training algorithm.

\[ W = \text{unif} \left( -\sqrt{\frac{6}{fan_{\text{in}} + fan_{\text{out}}}} , +\sqrt{\frac{6}{fan_{\text{in}} + fan_{\text{out}}}} \right) \]  

(3.2)

The binary cross-entropy loss function (BCE) was used as the function to be minimized during training. For GAZZETTA it was used as is, while for MULTITOX the final loss was calculated as the average of the result of BCE on each class. The BCE function is shown in Equation 3.3, where \( y \) is the true class of an example encoded as a binary digit, and \( p \) is the probability predicted by the model.

\[ \text{BCE}(p, y) = -(y \log(p) + (1 - y) \log(1 - p)) \]  

(3.3)

The Adaptive Moment Estimation optimization algorithm (ADAM) with learning rate 0.001 was used for training. The data were split in mini-batches of fixed size. In this optimization algorithm the weights are updated for each mini-batch using the gradient of the loss function normalized by estimates of the first and second moment of gradients calculated in previous mini-batches. Kingma and Ba (2015) showed that

\(^6\)https://radimrehurek.com/gensim/models/word2vec.html
\(^7\)https://fasttext.cc/
using ADAM leads to faster convergence compared to standard Stochastic Gradient Descend (SGD).

In order to prevent over-fitting, the early stopping method was used with patience 5, i.e. we stop training if the model has not improved on the Validation set in the last 5 epochs. The machine learning models in general tend to overfit the training set when the absolute value of the weights are becoming large, which is something that happens when we train for many iterations. Since we initialize the weights around zero, by stopping the training early we keep their absolute values small.

Following Pavlopoulos et al. (2017a), the evaluation was chosen to be the Area Under the Receiver Operating Characteristic curve (AUROC). The ROC curve is created by plotting the True Positive Rate (TPR, Equation 3.4) against the False Positive Rate (FPR, Equation 3.5) at various threshold settings. In case of MULTITOX the average AUROC of each class was used.

\[
\text{TPR} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (3.4)
\]

\[
\text{FPR} = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}} \quad (3.5)
\]

### 3.4 Baseline Model

In both datasets the models are compared to the state of the art model introduced by Pavlopoulos et al. (2017a). In their work they trained and evaluated their model on a previous version of the GAZZETTA dataset. They also used a different but similar English dataset from Wikipedia Talk pages. In their case the task was to predict whether a comment is toxic or not, in contrast to MULTITOX where the task is to also predict the type of toxicity that a comment contains. The model they introduced for toxicity detection is very similar to ATTRNN-m. The main difference is that they let the embeddings be altered during training, they didn’t use a projection layer, and in case of the English dataset, they initialized the Embedding Layer with embeddings pre-trained using the GLOVE algorithm (Pennington et al., 2014) on the Common Crawl corpus. Furthermore, in the case of the Greek dataset they used a subset of 100K examples for training during hyper-parameter tuning because training on the whole dataset was prohibitively slow.

### 3.5 Preliminary Experiments

The baseline was reimplemented (ATTRNN-PAV), the embeddings were kept fixed (ATTRNN-PAV-FE), a projection layer of size 128 with tanh as activation was introduced (ATTRNN-PAV-FE-PL), and in case of MULTITOX the GLOVE embeddings were replaced with ones that were trained using FASTTEXT on the Wikipedia Talk pages (ATTRNN-PAV-FE-PL-FT). These models use a unidirectional RNN-GRU of size 128, an attention MLP with 3 hidden layers of size 128 and logistic regression for the decision layer. The results of each model on the corresponding validation and development set can be found in Table 3.3 for MULTITOX and in Table 3.4 for GAZZETTA.

\[^8\text{See }\text{https://nlp.stanford.edu/projects/glove}\]
Chapter 3. Experiments

In the case of MULTITOX, by just freezing the embedding layer of the ATTRNN-PAV we notice an improvement, while introducing the projection layer also seems helpful. Furthermore, replacing the GLOVE embeddings with FASTTEXT ones boosts the performance even further.

In the case of GAZZETTA, we notice that freezing the embedding layer alone hurts the performance of the model, but by also using the projection layer the model surpasses the baseline. The important thing though, is that because the network has fewer trainable parameters, the model is trained five and a half times faster. By freezing the embeddings, we can now tune the model using the whole training set, something that was too time-consuming before.

Considering the results of these preliminary experiments, we kept the embeddings fixed and in the case of MULTITOX we replaced the GLOVE embeddings with FASTTEXT ones in subsequent experiments.

### 3.6 Hyper-Parameter Tuning

Each model (ATTRNN-m, MHATTRNN-m, CNN-m, RCNN-m) was tuned for 250 iterations for MULTITOX and for 100 iterations for GAZZETTA using Bayesian Optimization (Mukošus, 1975). In each iteration a model is trained on the training set and evaluated on the development set three times using a different randomization seed for the weight initialization. The score of an iteration is calculated as the average score of the three obtained. The hyper-parameters of each model alongside the choices that the tuner considered can be found in A. The tuner chose to use a projection layer in all the models, in both datasets. We used the Bayesian Optimization implementation of the BTB Python library.\(^9\)

### 3.7 Ensemble Models

Usually using a combination of models performs better than each single model by itself. A combination of models is called an ensemble and there are many ways to create it (Opitz and Maclin, 1999). The most common way to create it though is just to average predictions of many models. These models could either be entirely different

\(^9\)See [https://github.com/HDI-Project/BTB](https://github.com/HDI-Project/BTB)
with each other or could be different configurations of the same model. A benefit using this technique is that because we just average the predictions, we don’t induce any bias to the results, and as such we avoid over-fitting. This technique was only used for MULTITOX since the large size of the GAZZETTA dataset forbids us to train on it a lot of models. In MULTITOX, each model was retrained 80 times using different initial weights and a set of four predictions was created by averaging the predictions of each type of model separately (ATTRNN-ens, MHATTRNN-ens, CNN-ens, RCNN-ens). In addition, 20 trained models of each model type was chosen randomly in order to create a multi-model ensemble (ALL-ens). The reason for choosing only 20 models of each type in the latter ensemble is to ensure that all ensemble models use the same number of trained models.

### 3.8 Results

The results of each model on the corresponding validation, development and test set can be found in Table 3.5 for MULTITOX and in Table 3.6 for GAZZETTA.

In the case of MULTITOX, we observe that all models outperform the baseline. In addition, all ensemble models outperform the single ones, with ALL-ens having a slight edge. The models seem to achieve similar performance with the CNN-m performing slightly worse.

Looking closely to the results that each model achieved in each category separately (Figures 3.5–3.8), we can see that there isn’t a clear winner in almost any category. This was expected since the results were close and the models were tuned in order to perform best on average. One interesting observation though is that in the category Threat the attention based models are outperformed by the CNN based ones. The CNN-ens, which performs consistently worse in all other categories, compared to the rest ensemble models, achieves the best results in this one.

In case of GAZZETTA, again all the tuned models outperform the baseline. All the models perform similarly with the attention based models having a slight edge against the CNN based ones.

<table>
<thead>
<tr>
<th>Model Type</th>
<th>MULTITOX-VAL</th>
<th>MULTITOX-DEV</th>
<th>MULTITOX-TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>98.81 (0.05)</td>
<td>98.64 (0.09)</td>
<td>98.77 (0.05)</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>99.11 (0.03)</td>
<td>99.07 (0.02)</td>
<td>99.05 (0.02)</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>99.13 (0.02)</td>
<td>99.10 (0.05)</td>
<td>99.07 (0.01)</td>
</tr>
<tr>
<td>CNN-m</td>
<td>99.09 (0.02)</td>
<td>99.04 (0.03)</td>
<td>99.02 (0.02)</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>99.13 (0.02)</td>
<td>99.14 (0.02)</td>
<td>99.08 (0.02)</td>
</tr>
<tr>
<td>ATTRNN-ens</td>
<td>99.20</td>
<td>99.18</td>
<td>99.14</td>
</tr>
<tr>
<td>MHATTRNN-ens</td>
<td>99.21</td>
<td>99.18</td>
<td>99.15</td>
</tr>
<tr>
<td>RCNN-ens</td>
<td>99.22</td>
<td>99.21</td>
<td>99.17</td>
</tr>
<tr>
<td>ALL-ens</td>
<td>99.26</td>
<td>99.23</td>
<td>99.18</td>
</tr>
</tbody>
</table>

**Table 3.5: MULTITOX AUROC Results.**
Chapter 3. Experiments

Table 3.6: GAZZETTA AUROC Results.

<table>
<thead>
<tr>
<th>Model</th>
<th>GAZZETTA-VAL</th>
<th>GAZZETTA-DEV</th>
<th>GAZZETTA-TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>81.49 (0.05)</td>
<td>80.31 (0.03)</td>
<td>80.46 (0.05)</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>82.11 (0.05)</td>
<td>80.64 (0.06)</td>
<td>81.11 (0.05)</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>82.11 (0.03)</td>
<td>80.73 (0.04)</td>
<td>81.14 (0.05)</td>
</tr>
<tr>
<td>CNN-m</td>
<td>81.97 (0.06)</td>
<td>80.51 (0.07)</td>
<td>80.98 (0.09)</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>82.05 (0.04)</td>
<td>80.55 (0.03)</td>
<td>81.03 (0.04)</td>
</tr>
</tbody>
</table>

Figure 3.5: MULTITOX-DEV AUROC per Category (Single Models).
Chapter 3. Experiments

Figure 3.6: **MULTITOX-TEST AUROC per Category (Single Models).**

Figure 3.7: **MULTITOX-DEV AUROC per Category (Ensemble).**
Chapter 3. Experiments

Figure 3.8: MULTITOX-TEST AUROC per Category (Ensemble).

Figure 3.9: GAZZETTA AUROC Results.
Chapter 3. Experiments

3.9 Highlighting Suspicious Words

A nice feature that the attention based models have is the ability to highlight suspicious words. This can be done by using the attention scores provided as the “suspiciousness” score for each word. In this way we are getting a justification about the model’s predictions. This is especially helpful for moderators, since they need to decide quickly about whether a comment should be rejected or not.

If we are dealing with a multi-label task like MULTITOX, we would like to be able to highlight the words of a comment in many ways, one for each class. In order to do that, another model was introduced, which was named Multi-Output Multi-Attention RNN (MOMATRNN-m). This model is similar to ATTRNN-m, but it uses 6 attention and output layers, 1 for each category. The model achieved comparable performance to the ATTRNN-m (MULTITOX-VAL: 99.13, MULTITOX-DEV: 99.09, MULTITOX-TEST: 99.05) while not tuned. Instead the same hyper-parameters were used as in ATTRNN-m. With this model we can highlight the words of a comment differently for each class, by using the attention scores produced by each attention layer.

Figure 3.10 depicts a toxic comment repeated 7 times. The words of the first one were highlighted based on the attention scores produced by the ATTRNN-m, while the words of the others were highlighted based on the attention scores produced by the MOMATRNN-m for each category.

As we can see, the attention scores given by MOMATRNN-m are very specific for each category. For example, for the category Threat all the attention is gathered in the word “kill” in contrast to the category Identity Hate where all the attention is gathered in the word “Jewish”.

Randomly chosen examples from MULTITOX-DEV that contain at least one type of toxicity, alongside the attention scores produced by MOMATRNN-m can be found in B.
Chapter 4

Related Work

Recently, the problem of abusive content detection has drawn a lot of scientific attention. Proof of this increased interest of the community is the creation of the 1st Abusive Language Workshop\(^1\). In addition, Kaggle’s Toxic Comment Classification Challenge gathered 4,551 teams.\(^2\) Until recently, most of the methods used for this task were based on hand-crafted features. Now, the trend seems to change direction, with deep learning models gaining the upper hand since large annotated datasets are being created and made publicly available.

Wulczyn et al. (2017) created and experimented with three datasets for abusive content detection; the Personal Attack dataset where 115K comments from Wikipedia Talk pages annotated as containing personal attack or not, the Aggression dataset where the same comments were annotated as containing aggression or not, and the Toxicity dataset that includes 159K comments again from Wikipedia Talk pages, were annotated as being Toxic or not. They compared the performance of Logistic Regression (LR) and an MLP using \(n\)-grams of words or \(n\)-grams of characters as features. They showed that the best performing model was the MLP operating on top of \(n\)-grams of characters.

Pavlopoulos et al. (2017a) showed that RNN based methods surpassed in performance all models considered by Wulczyn et al. (2017) in the same datasets. The RNN that used a self-attention mechanism is used as baseline in this thesis. They also introduced the Greek dataset GAZZETTA, an extended version of which was used in this thesis. The new version of this thesis uses all the comments of the previous one as training examples and approximately 814K additional comments for evaluation. They also introduced a way to tune the decision threshold of a model depending on the desired coverage. By coverage we refer to the approximate percentage of comments that we want the system to handle by itself and can be altered on the fly depending on the moderators’ current workload. In subsequent work, Pavlopoulos et al. (2017c) showed that using additional user information in the form of trainable user embeddings boosts the performance significantly.

Djuric et al. (2015) experimented with a dataset containing 56K comments that include hate speech alongside 895K clean comments from the Yahoo Finance website. They used PARAGRAPH2VEC (Le and Mikolov, 2014) for joint modeling of comments and words. Using PARAGRAPH2VEC they were able to create distributed representations for the comments in an unsupervised manner. Then, they showed that using these representation as input to an LR classifier achieves better results than the usual \(tf\) and \(tf-idf\) features.

Nobata et al. (2016) experimented with a dataset containing abusive and clean comments posted on Finance and News articles of the Yahoo! website. They considered \(n\)-gram features (character \(n\)-grams, token unigrams and bigrams), linguistic

---

\(^1\)See https://sites.google.com/site/abusivelanguageworkshop2017/

\(^2\)See https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
features (length of comment, average word length, number of punctuation, etc.), syntactic features produced by a dependency parser, and distributed representation produced by algorithms like Word2Vec and Paragraph2Vec. They evaluated models that used different subsets of such features and concluded that using all features achieved the best results. They also compared their model with the one proposed by Djuric et al. (2015) and showed that the additional hand-crafted features helped. They also experimented with a temporal dataset from the same domain. Specifically, they trained their model to a chronologically old but big dataset and compared the results to those achieved when the model is trained on data that are more chronologically close to the evaluation set. They showed that training on recent data gives an advantage. This also justifies the fact that the Gazzetta dataset was split in such a way so that the training set is older than the evaluation sets.

Davidson et al. (2017) collected 25K tweets and asked from CrowdFlower annotators to label them as being in one of the following categories: “hate speech”, “offensive but not hate speech”, or “neither offensive nor hate speech”. Their best results were obtained using an LR classifier with hand-crafted features as input. Some of them were: token statistics (tf-idf), part-of-speech statistics, a sentiment lexicon designed for social media, Twitter specific features like number of mentions and number of retweets.

Waseem and Hovy (2016) aimed to detect hate speech, experimenting with 16k tweets that they annotated themselves. They analyzed the impact of various linguistic, demographic and geographic features in conjunction with character n-grams on the performance of an LR classifier. In subsequent work (Waseem, 2016), they compared the results of a LR classifier using several hand-crafted features on a Twitter dataset that was annotated twice; once by experts and a second time by amateur annotators recruited from CrowdFlower. They found that amateur annotators are more likely than expert ones to label items as hate speech, and that systems trained on expert annotations outperform systems trained on amateur ones.

3https://www.figure-eight.com
Chapter 5

Conclusions

In this thesis, we discussed and compared models to tackle the task of toxicity detection in two datasets: the GAZZETTA dataset, which corresponds to the binary classification task of detecting abusive content in user comments, and the MULTITOX dataset which corresponds to the multi-label classification task of detecting what types of toxicity, if any, a comment contains. The RNN with an attention mechanism (ATTRNN-PAV), which was the previous state of the art for abusive content detection, was reimplemented using the proposed architecture, and was considered the baseline. In the case of MULTITOX, freezing the embedding layer, adding a projection layer and replacing the GLOVE embeddings with FASTTEXT ones boosted the performance of the model. In the case of GAZZETTA, by freezing the embedding layer alone we hurt the performance, but we surpassed the baseline by also introducing the projection layer. Furthermore, keeping the embeddings fixed resulted in significant reduction in training time, enabling us to tune the models using the whole training dataset in the case of GAZZETTA, something that had never been done before. Tuning the tweaked version of the attention RNN (ATTRNN-m) resulted in an additional improvement. Next, the other deep learning models were created and tuned (MHATTRNN-m, CNN-m, RCNN-m). All the tuned models outperformed the baseline but performed similarly to each other. The CNN-m performed slightly worse in the case of MULTITOX while the attention based models performed slightly better than the CNN based ones in the case of GAZZETTA. The ensemble models used in the case of MULTITOX surpassed all single type models, with the ALL-ens ensemble having a slight edge. Finally, it was shown that we can use a multi-attentional model (MOMATTRNN-m) to highlight the suspicious words of a comment for each category.

5.1 Future Work

An idea for future work is to divide the categories of MULTITOX into non-overlapping sets and create models that are tuned and trained in these sets. In the extreme case each set would contain only one category. But it would be interesting to see if it is helpful for a model to be trained on a combination of categories. One way these combinations could be chosen is based on the correlation among the categories in the training set.

We observed that word embeddings trained with the FASTTEXT algorithm helped in the case of MULTITOX. Since FASTTEXT uses character n-grams, maybe it would be also helpful to use a module that utilizes sub-word information when creating the input representation. An example would be to use an RNN (Ling et al., 2015) or a CNN (Zhang et al., 2015) encoder that reads the characters of each word to produce a dense representation. This dense representation can be then concatenated to the corresponding word embedding. Finally, the idea of pre-training an RNN for language modeling on an unlabeled corpus and using it for generating additional information
about the input of a subsequent task (Howard and Ruder, 2018) could also be useful for the task of toxicity detection.
Bibliography


Appendix A

Model Architectures

Table A.1 describes the hyper-parameters, Table A.2 describes their possible values, Tables A.3–A.7 and Tables A.8–A.12 contain the values that the tuner chose for MULTI-TOX and GAZZETTA respectively, alongside the architecture of the baseline (ATTRNN-PAV).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>The batch size.</td>
</tr>
<tr>
<td>pl size</td>
<td>The size of the projection layer.</td>
</tr>
<tr>
<td>pl act</td>
<td>The activation function of the projection layer.</td>
</tr>
<tr>
<td>pl seq dp</td>
<td>Sequence Dropout of the projection layer.</td>
</tr>
<tr>
<td>rnn size</td>
<td>The size of the RNN.</td>
</tr>
<tr>
<td>rnn depth</td>
<td>The depth of the RNN.</td>
</tr>
<tr>
<td>rnn bid</td>
<td>Whether the RNN is bidirectional.</td>
</tr>
<tr>
<td>rnn seq dp</td>
<td>Sequence Dropout of the RNN.</td>
</tr>
<tr>
<td>cnn fs</td>
<td>Filter sizes of the CNN.</td>
</tr>
<tr>
<td>cnn fn</td>
<td>Number of filters for each filter size of the CNN.</td>
</tr>
<tr>
<td>cnn act</td>
<td>The activation used after each convolution of the CNN.</td>
</tr>
<tr>
<td>cnn dp</td>
<td>The Dropout used after each convolution of the CNN.</td>
</tr>
<tr>
<td>att mlp depth</td>
<td>Number of hidden layer of the attention MLP.</td>
</tr>
<tr>
<td>att mlp size</td>
<td>Size of the hidden layers of the attention MLP.</td>
</tr>
<tr>
<td>att mlp act</td>
<td>Activation function of the attention MLP.</td>
</tr>
<tr>
<td>mhatt heads</td>
<td>Number of heads of the MHATTRNN-m.</td>
</tr>
<tr>
<td>out mlp depth</td>
<td>Number of hidden layer of the output MLP.</td>
</tr>
<tr>
<td>out mlp size</td>
<td>Size of the hidden layers of the output MLP.</td>
</tr>
<tr>
<td>out mlp act</td>
<td>Activation function of the output MLP.</td>
</tr>
</tbody>
</table>

Table A.1: Description of the Hyper-Parameters.
### Table A.2: Choices that the Tuner Considered for the Hyper-Parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>[128, 256, 512, 768, 1024]</td>
</tr>
<tr>
<td>pl size</td>
<td>[None, 128, 256]</td>
</tr>
<tr>
<td>pl act</td>
<td>[tanh, relu, prelu]</td>
</tr>
<tr>
<td>pl seq dp</td>
<td>[None, 0.25, 0.5]</td>
</tr>
<tr>
<td>rnn size</td>
<td>[128, 256]</td>
</tr>
<tr>
<td>rnn depth</td>
<td>[1, 2]</td>
</tr>
<tr>
<td>rnn bid</td>
<td>[True, False]</td>
</tr>
<tr>
<td>rnn seq dp</td>
<td>[None, 0.25, 0.5]</td>
</tr>
<tr>
<td>cnn fs</td>
<td>([2, 3], [2, 3, 4], [2, 3, 4, 5])</td>
</tr>
<tr>
<td>cnn fn</td>
<td>[128, 256]</td>
</tr>
<tr>
<td>cnn act</td>
<td>[relu, prelu]</td>
</tr>
<tr>
<td>cnn dp</td>
<td>[None, 0.25, 0.5]</td>
</tr>
<tr>
<td>att mlp depth</td>
<td>[0, 1, 2, 3]</td>
</tr>
<tr>
<td>att mlp size</td>
<td>[128, 256]</td>
</tr>
<tr>
<td>att mlp act</td>
<td>[128, 256]</td>
</tr>
<tr>
<td>att mlp act</td>
<td>[relu, prelu]</td>
</tr>
<tr>
<td>mhatt heads</td>
<td>[2, 4, 8, 16, 32]</td>
</tr>
<tr>
<td>out mlp depth</td>
<td>[0, 1, 2, 3]</td>
</tr>
<tr>
<td>out mlp size</td>
<td>[128, 256]</td>
</tr>
<tr>
<td>out mlp act</td>
<td>[relu, prelu]</td>
</tr>
</tbody>
</table>

### Table A.3: Model Architectures for MultiTox (1/5).

<table>
<thead>
<tr>
<th>ATTRNN-PAV</th>
<th>ATTRNN-m</th>
<th>MHATTRNN-m</th>
<th>CNN-m</th>
<th>RCNN-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>128</td>
<td>512</td>
<td>768</td>
<td>768</td>
</tr>
<tr>
<td>emb alg</td>
<td>Glove</td>
<td>Fasttext</td>
<td>Fasttext</td>
<td>Fasttext</td>
</tr>
<tr>
<td>pl size</td>
<td>-</td>
<td>256</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>pl act</td>
<td>-</td>
<td>relu</td>
<td>relu</td>
<td>prelu</td>
</tr>
<tr>
<td>pl seq dp</td>
<td>-</td>
<td>0.5</td>
<td>0.25</td>
<td>0.5</td>
</tr>
</tbody>
</table>

### Table A.4: Model Architectures for MultiTox (2/5).

<table>
<thead>
<tr>
<th>ATTRNN-PAV</th>
<th>ATTRNN-m</th>
<th>MHATTRNN-m</th>
<th>CNN-m</th>
<th>RCNN-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnn size</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>rnn depth</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>rnn bid</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>rnn seq dp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.25</td>
</tr>
</tbody>
</table>
Appendix A. Model Architectures

<table>
<thead>
<tr>
<th>cnn fs</th>
<th>cnn fn</th>
<th>cnn act</th>
<th>cnn dp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CNN-m</td>
<td>[2, 3]</td>
<td>256</td>
<td>relu</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>[2, 3]</td>
<td>128</td>
<td>relu</td>
</tr>
</tbody>
</table>

**Table A.5:** Model Architectures for MultiTox (3/5).

<table>
<thead>
<tr>
<th>att mlp depth</th>
<th>att mlp size</th>
<th>att mlp act</th>
<th>mhatt heads</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>3</td>
<td>128</td>
<td>relu</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>4</td>
<td>128</td>
<td>prelu</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>1</td>
<td>128</td>
<td>prelu</td>
</tr>
<tr>
<td>CNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table A.6:** Model Architectures for MultiTox (4/5).

<table>
<thead>
<tr>
<th>out mlp depth</th>
<th>out mlp size</th>
<th>out mlp act</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>2</td>
<td>128</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>CNN-m</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>2</td>
<td>128</td>
</tr>
</tbody>
</table>

**Table A.7:** Model Architectures for MultiTox (5/5).

<table>
<thead>
<tr>
<th>batch size</th>
<th>emb alg</th>
<th>pl size</th>
<th>pl act</th>
<th>pl seq dp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>128</td>
<td>w2v</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>1024</td>
<td>w2v</td>
<td>256</td>
<td>tanh</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>512</td>
<td>w2v</td>
<td>256</td>
<td>tanh</td>
</tr>
<tr>
<td>CNN-m</td>
<td>512</td>
<td>w2v</td>
<td>256</td>
<td>prelu</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>512</td>
<td>w2v</td>
<td>256</td>
<td>tanh</td>
</tr>
</tbody>
</table>

**Table A.8:** Model Architectures for Gazzetta (1/5).

<table>
<thead>
<tr>
<th>rnn size</th>
<th>rnn depth</th>
<th>rnn bid</th>
<th>rnn seq dp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>128</td>
<td>1</td>
<td>False</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>128</td>
<td>2</td>
<td>True</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>256</td>
<td>1</td>
<td>False</td>
</tr>
<tr>
<td>CNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>128</td>
<td>1</td>
<td>True</td>
</tr>
</tbody>
</table>

**Table A.9:** Model Architectures for Gazzetta (2/5).
### Table A.10: Model Architectures for Gazzetta (3/5).

<table>
<thead>
<tr>
<th>Model</th>
<th>att mlp depth</th>
<th>att mlp size</th>
<th>att mlp act</th>
<th>mhatt heads</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>3</td>
<td>128</td>
<td>relu</td>
<td>-</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>2</td>
<td>256</td>
<td>relu</td>
<td>-</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>3</td>
<td>128</td>
<td>relu</td>
<td>32</td>
</tr>
<tr>
<td>CNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table A.11: Model Architectures for Gazzetta (4/5).

<table>
<thead>
<tr>
<th>Model</th>
<th>out mlp depth</th>
<th>out mlp size</th>
<th>out mlp act</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRNN-PAV</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATTRNN-m</td>
<td>1</td>
<td>128</td>
<td>relu</td>
</tr>
<tr>
<td>MHATTRNN-m</td>
<td>3</td>
<td>256</td>
<td>relu</td>
</tr>
<tr>
<td>CNN-m</td>
<td>3</td>
<td>128</td>
<td>relu</td>
</tr>
<tr>
<td>RCNN-m</td>
<td>3</td>
<td>128</td>
<td>prelu</td>
</tr>
</tbody>
</table>

### Table A.12: Model Architectures for Gazzetta (5/5).
Appendix B

Examples of Highlighting Abusive Words

Figures B.1–B.4 show the attention scores produced by the MOMATTRNN-m model for four randomly chosen examples from MULTITOX-DEV that contain at least one type of toxicity.

Figure B.1: Highlighting Suspicious Words (1/4).
Appendix B. Examples of Highlighting Abusive Words

Figure B.2: Highlighting Suspicious Words (2/4).

Figure B.3: Highlighting Suspicious Words (3/4).
Figure B.4: Highlighting Suspicious Words (4/4).