Radboud University Nijmegen

Faculty of Science

Sentence Level Event Classification in Newspaper Articles

Master's Thesis

Author:
Simge Ekiz Başar

Supervisors:
Faegheh Hasibi
Ali Hürriyetoğlu

Second reader:
Djoerd Hiemstra

February, 2020

Abstract

Processing online news media content can provide a significant amount of information and knowledge. Due to the high volume of online news, it is nearly impossible to process and analyse these unstructured texts with human power. The human effort can be scaled down by automatically reducing the large volume of text into relevant information pieces. In this study, we focus on developing a method for automatically detecting sentences that contain specific event information in news articles. We propose and make comparisons between two text classification approaches: context-agnostic classification and context-dependent classification. For both of the approaches, we employ two state-of-the-art pre-trained contextual word embeddings, namely ELMo and BERT. We observe that the BERT-based context-agnostic neural model outperform baseline approaches and also other models, achieving 82.2% F1 score.

Contents

Chapter 1

Introduction

The rise of the web rapidly changed the way of communication and how the information is created, spread and used. It caused an exponential increase in digital data. Unquestionably, processing such data provides a significant amount of information and knowledge. However, most of the information on the web is written in natural languages, and thus, stored as unstructured text. Due to the high volume and nature of the unstructured text, it is nearly impossible to process and analyse these unstructured texts with human power. As a result, automatically processing and analysis of information from unstructured texts is of great importance and has become a popular research area [48, 13]. The recent developments in text mining and artificial intelligence made this automation applicable to any domain.

The goal of this research is identifying the sentences that contain specific information in news articles. Reducing the large volume of text into relevant information pieces can scale down the human effort during further analysis. Further automation of manual analysis can include creating a structured database, which is called information extraction [40]. Processing relevant sentences exclusively can reduce the false positives in the downstream information extraction task for event detection, which would enhance the reliability of the event database.

This thesis is part of a multidisciplinary research work package that aims to help social scientists understand global problems better by monitoring online media sources and creating a sociopolitical event database. Hence, in this study, our aim is automatically detecting the sentences that contain information about protest-related events. We are willing to answer the question *"How can we identify event-related sentences?"*. We approach the issue as a text classification problem, where our goal is to assign each sentence a predefined category automatically.

1.1 Objectives

Context-agnostic sentence classification with contextual word embeddings.

Our first objective in this study is making use of contextual word embeddings to improve performance of sentence level event classification. Specifically, our first research question is *"How can we incorporate pre-trained embeddings to perform sentence classification?"*.

With the promising developments in pre-trained word embeddings, it is worth to explore current state-of-the-art word embeddings for our sentence classification task. Recently, contextual word embeddings have shown potential improvements for various NLP tasks by encoding word, based on their context.

To answer this research question, we propose a method that incorporates the latest state-of-the-art contextual word embeddings, namely ELMo [32] and BERT [10], to an Artificial Neural Network (ANN) model. We compare our method with TF-IDF based representation of sentences and use various classical machine learning algorithms, such as Decision Trees, Random Forests, and Support Vector Machines. We show that ANN architecture with BERT embedding outperforms the other models. Finally, we conclude that the contextual word embeddings improves the performance of the sentence classification in a context-agnostic setup.

Context-dependent sentence classification.

Our second objective is taking the preceding and the following information of a sentence into account to understand the meaning of a sentence. We hypothesize that it is essential to process the context when we classify sentences. Thus, we are willing to answer the question *"Does taking the context of a sentence into account improves sentence level event classification?"*.

To answer this question, we propose a Recurrent Neural Network (RNN) architecture that can classify sequences of sentences that are encoded with contextual word embedding models. We made comparisons to the aforementioned context-agnostic methods. We observed that the context-dependent network with ELMo embeddings performs as good as ELMo embeddings in the context-agnostic setup. However, the context-agnostic network with BERT embeddings is still the best performing approach. We conclude that the context-dependent approach is still worth to explore in further studies, although it does not immediately improve the classification performance.

Outline

The rest of this thesis is organized as follows. In Chapter 2, we cover previous work that has been done for text classification tasks and briefly describe the methods we used and some underlying concepts. In Chapter 3, we mention the usage of these methods in our text classification use case. We present approaches and how we opt for our decisions. In Chapter 4, we describe the dataset, discuss the evaluation metrics and present our results. In Chapter 5, we draw conclusions from our work and answer the research questions and discuss further improvements.

Chapter 2

Related Work

2.1 Text Classification

Text classification is one of the fundamental tasks of Natural Language Processing (NLP). Ultimately, we want computers to analyse an input text and extract an overall meaning or a specific knowledge reliably. Text classification tasks partly achieve this by categorising and sorting textual data based on the information in its content. In a broad sense, text classification is the process of automatically assigning a predefined class to a piece of natural language text [39].

Automatically classifying (also referred to as labeling or categorising) text can be used in a broad range of applications, and it is useful in almost any domain that interacts with textual data. For instance, search engines use text classification approach to retrieve information relevant to the topics [53]. Companies can monitor business trends from various textual sources which help them to make reliable business decisions. Online sellers can see the overall user opinion by categorising product reviews as positive or negative with sentiment analysis. By analysing newspapers or social media, authorised people can take precautions for critical situations (such as natural disasters) or take actions in the time of need.

Early methods of text classification were based on manually programming linguistic rules. This technique, known as knowledge engineering, was the dominant approach in its era [28]. However, it requires extensive analysis, comprehensive testing and in-depth expert knowledge for each domain [41]. Hand-crafting, maintaining and updating the rules are time-taking and complicated, as the rules can interfere with each other. The difficulties in customisation and generalisation of these algorithms also harm their scalability [51].

More recently, machine learning-based text classification approaches have become an adequate alternative solution, instead of relying on predefined rules [41]. Machine learning models can learn from past observations and capture insights from data. Thus, given some amount of observed examples, a machine learning algorithm learns associations between its input texts and its output by itself [26]. This ability shifted the engineering effort of hand-crafting custom rules (classifiers) to teaching statistical learning algorithms by feeding them example instances [41].

A common approach to machine learning-based text classification is generating a language model by using the features extracted from the text itself. We can divide the process of a text classifier learning into two steps [36]. The first step is feature extraction, where we select the features of the text that will be used to generate a numerical representation of the text. The features can vary depending on the classification task, the subject domain, or the methodology of the algorithm. The feature list can be a bag-of-words (e.g., using TF-IDF) or distributed representations of words and sentences. The second step is the learning, where a statistical algorithm is trained to assign labels to text by automatically analysing the numerical representations and the given labels. The machine learning algorithms at the core of the learning are essentially mathematical formulas, and they can be used regardless of the subject domain. Some of the well-known machine learning algorithms such as decision trees, random forests, support vector machines, and artificial neural networks are explained in Sections 2.2 and 2.3.

Classification of text can be divided into four different levels; document, paragraph, sentence, and sub-sentence levels [20]. The most well-known version of text classification and the first application that comes to mind is classifying entire documents such as newspapers and web pages. Despite being widely studied, document classification still preserves its challenges due to the unstructured nature of the text. Thus, it is still an open research area. Recent studies incorporate artificial neural networks and features extracted from both text and images attached to the documents [3].

Sentence classification and sub-sentence classification tasks are studied under the short-text classification as their contents are shorter than a regular document. Analysing short text came under the spotlight with the rise of the Web 2.0 that introduced an interactive internet model where people participate by sharing information in forms of small texts such as comments, reviews, and tweets [36]. Due to the amount of content that it can contain, sentences are also accepted as short texts.

Processing short text introduces new challenges to text classification. Especially feature extraction is challenging because a short text is low on content. First, the lack of content causes the feature sets to become too sparse. The feature sparsity reduces the performance of the classical machine learning algorithms [42]. Second, the lack of content makes it harder to discover powerful linguistic features such as co-occurred words [24]. Thus, the similarity algorithms based on such features are not performing well with short texts. Processing Web 2.0 short text is challenging due to the grammatical errors and the dynamic nature of the Internet [24]. However, these are not the cases with processing the sentences in our use case, since the sentences are usually parts of well-written documents such as newspapers.

To overcome the challenges in feature extraction, researchers have proposed ways to enhance the feature set by using external knowledge. A study proposed a method that made use of Latent Dirichlet Allocation (LDA) to create a feature thesaurus [46], and another has shown improvements by adding user-defined categories and hidden topics in Wikipedia articles into the knowledge base [33]. Another study used domain-specific features and author profiling to increase the performance of a tweet classification algorithm [44]. More recently, developments in artificial neural network-based word embedding methods have shown success on semantic encoding. With word embeddings, the granular data gain the ability to hold more information implicitly. As a result, the recent studies used the word embeddings that are generated from large datasets as their features. It is shown that encoding short text with these word embeddings and using them as features for artificial neural networks achieves state-of-the-art performance [47, 19, 50].

To our knowledge, task-wise, the closest study to ours is [29]. Naughton et al. have approached the event detection task as a binary classification of the sentences. They used support vector machines algorithm to train their model based on a feature set consisting of specific linguistic features such as the terms, part-of-speech and noun chunks. Given that the study is from 2010, it could not involve any of the recent classification techniques we present in our study. Thus, we can see our study as a methodological update over the work has been done on sentence level event classification.

Method-wise, we see that the closest study to ours is [23]. This method depends on the idea that the short texts are next to each other in a broader context, similar to our context-dependent approach. The method is based on Recurrent Neural Networks(RNN) that takes a short text and its context into account. Their architecture starts with feeding word embeddings into a neural network layer and applying max-pooling to encode the sentences individually. As of word embeddings, their choice was word2vec [25] model pre-trained on Google News and GloVe[31] model pre-trained on Twitter stream. Both of the word embedding models are well-known feature-based embeddings, whereas we use the recently developed embeddings that are either contextual or fine-tunable (see Section 2.4). They have also introduced a hyperparameter, named history size, that sets the length of the sequences, similar to our window-width parameter used in the sliding window strategy. As a difference, we also introduce a model that uses the full article as context.

2.2 Classical Machine Learning

In this section, we explain some of the supervised classical machine learning algorithms that are formerly known as the state-of-the-art techniques for text classification tasks. These approaches are used in many applications and research studies, including but not limited to short text topic classification, semantic text analysis, and multimodal classification [2]. In recent studies, we see that these algorithms are outperformed by artificial neural network techniques, even though the interpretability of the results is sacrificed [20]. Nowadays, classical machine learning algorithms are used by many researchers in their studies as a baseline system to understand the capacity of their proposed approaches [49].

2.2.1 Decision Tree

Decision Tree (DT) algorithm discovers a set of rules by analysing a labeled dataset. Then, the algorithm uses the rules to predict unseen data. These discovered rules are sorted to form a tree where each node represents a decision point, and each leaf contains an output class (see Figure 2.1). In the context of machine learning, the decision trees are introduced with the Iterative Dichotomiser 3 (ID3) algorithm [35], which works with categorical data used in classification tasks. Then, ID3 is followed by successors such as C4.5 and CART.

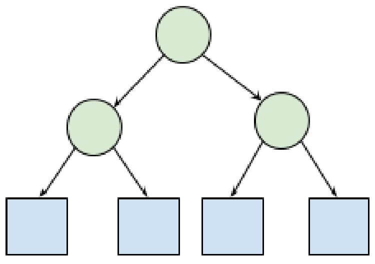


Figure 2.1: A visualization of decision tree algorithm. Each node is a decision point and each leaf represents an outcome.

There are various implementations of DT for different conditions and tasks. Nevertheless, they all follow the same tree metaphor. At the training process, the algorithm calculates the information gain for each feature. Then it selects the feature with the most significant gain as the root (first node) of the tree. Next, it continues to form the tree by partitioning the data at each selected feature. This process is repeated for each branch. When a subset of examples of the same class is formed, the tree places the class as the leaf (final node) of that branch. It can also stop when there are no more features to split even though the subset contains a variety of classes. In that case, the algorithm again places a tree, but this time the most common class in the subset is set as the leaf's value.

In some variants of DT, another splitting criteria called Gini impurity is calculated in order to decide on the node to split, instead of information gain. Gini impurity is a calculation of misclassification chance when a label is randomly assigned based on the distribution of the labels in the set. There is not a statistically significant difference in the results of using information gain and Gini impurity [6]. However, Gini impurity has less computational complexity since it avoids computing a logarithmic function, unlike the information gain.

At the prediction, when a new data point with a new feature set is introduced, the algorithm starts from the feature node at the top of the tree. At each node, the algorithm decides whether it should continue going down the tree from the left or the right set of rules, based on the value in the corresponding feature point in the data. Finally, the algorithm stops when it reaches a leaf and returns the value of the leaf as the outcome.

Because of its straightforward implementation, it is possible to visualize the rules discovered by DT algorithms and interpret the results by understanding the reasons behind them. Contrarily, as a disadvantage of this simplicity, DT algorithms are not successful at complex tasks that require large vector spaces. It has been shown that having an imbalanced dataset or having too many nodes on a decision tree can cause overfitting [37]. Overfitting is a generalization problem where the model shows high performance on the training dataset, while it is not effective in processing the unseen test dataset.

2.2.2 Random Forest

Random Forests (RF) algorithm advances over the same tree metaphor used in the Decision Tree algorithms. It operates as an ensemble method that consists of a large number of individual decision tree models. Each decision tree model produces a classification result. In the end, the class returned by the majority of the Decision Tree models becomes the prediction of the Random Forest model [5]. This process is called majority voting and demonstrated in Figure 2.2.

At the training time, the Random Forest algorithm uses bootstrap aggregating (a.k.a bagging) to distribute the instances to the Decision Tree models. For each Decision Tree model, it randomly samples a number of instances from the entire dataset without replacement. As a result, there are no pairs of Decision Tree models that take the same subset of instances. Owing to this, we are forcing the trees to have different node splits, different rules and different decision-making processes. Random Forest algorithm also distributes features to the Decision Tree models by using random subspace method. To train each Decision Tree model, it randomly samples a subset of features with replacement, instead of giving the entire feature set [4]. Thus, the models may have overlapping feature sets, although there is a low probability that they will be the same when a sufficient number of features are given. This process of feature sampling forces Decision Tree models to have different decision-making processes further.

As a result of the bootstrap aggregating and random subspace method, the correlation between the Decision Tree models is minimized. Training each model on a different set of data and with a different set of features reduces the risk of bias towards a particular instance type or a subset of features, and eventually reducing the risk of overfitting for the Random Forest model. Even though a Decision Tree model in the ensemble may overfit on a single data point by itself, it is corrected by the wisdom of the crowd during the majority voting process. This case makes the Random Forest algorithm produce fewer errors when compared to individual Decision Tree algorithms.

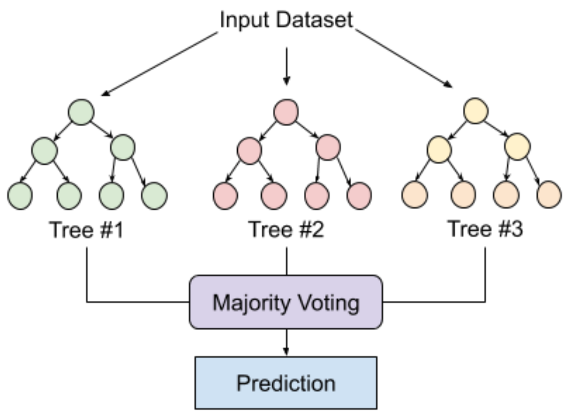


Figure 2.2: A visualization of Random Forest algorithm.

2.2.3 Support Vector Machines

Support Vector Machines (SVM) algorithm takes the feature set of each instance as a vector and places it into a vector space. At the training time, given a dataset with labeled instances, SVM forms hyperplanes to separate the data based on their classes. The aim of the SVM algorithm is calculating the optimal hyperplane that maximizes the margin between the classes [9]. At the prediction, the new instances are placed into the same vector space, and their labels are decided based on their position to the hyperplane.

Figure 2.3 visualizes how labeled instances are separated with a single hyperplane into their respective groups by the SVM algorithm. The figure shows a simplified version by taking each instance as a 2-dimensional point. In a real application, however, the data is represented with long feature vectors, and every feature of the vector adds another dimension. Primarily the text classification tasks require having high dimensional vector spaces in order to have useful representations.

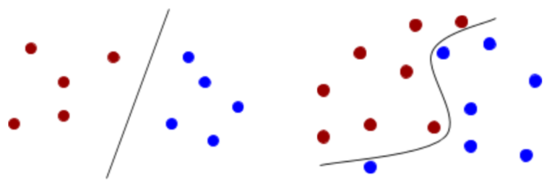


Figure 2.3: Hyperplanes are separating datasets. In this visualization, each instance has two features, and so, appearing as points on the coordinate system. The graph on the left shows a linearly separable dataset, and the one on the right shows a non-linearly separable dataset.

Initially, the SVM algorithm separates the data with a linear hyperplane. However, every dataset has different characteristics, and so, different feature sets. Different feature sets may require different shapes of hyperplanes. So, it is also likely to see datasets that cannot be linearly separated. Each object in a non-linearly separable vector space is transformed into a higher dimension, where the linear separation of the classes is possible (shown in Figure 2.4) [15]. This transformation is known as the kernel trick and done with the help of specific mathematical formulas named kernel functions.

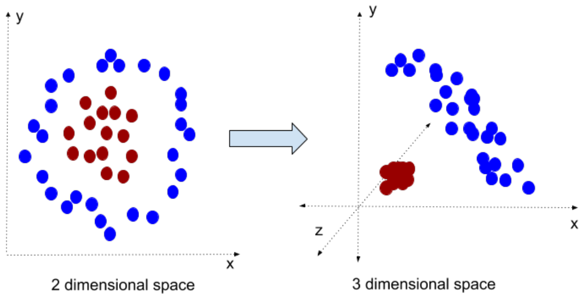


Figure 2.4: Kernel trick representation; transformation of the feature vector space into a higher dimension with the help of the kernels to be able make it linearly separable.

Given that ϕ is the feature mapping, X_i is the support vector, and X_j is a multi-dimensional real vector, a generalized definition of the kernel function can be written as:

$$K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j) \tag{2.1}$$

In order to make an optimal separation, different kernel methods are introduced. Some of the well-known types of kernels are linear, polynomial, radial basis function (RBF) and sigmoid funcitons [34]. Given that C and γ are constant parameters, and d is the degree, the equations of these kernels respectively:

$$K_{linear}(X_i, X_j) = X_i \cdot X_j \tag{2.2}$$

$$K_{polynomial}(X_i, X_j) = (\gamma \cdot X_i \cdot X_j + C)^d \tag{2.3}$$

$$K_{RBF}(X_i, X_j) = \exp(\gamma \mid X_i - X_j \mid^2) \tag{2.4}$$

$$K_{sigmoid}(X_i, X_j) = \tanh(X_i \cdot X_j + C)^d \tag{2.5}$$

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are computational graphs designed to model the way the human brain works. They consist of layers of nodes that hold a real number. The first layer of an ANN is referred to as the input layer, and the last layer is called the output layer. The layers between them are called hidden layers. When a neural network contains a single hidden layer, it is referred to as a shallow network. If there is more than one hidden layer, it is considered a deep neural network.

The nodes are assumed to be the neurons in the human brain. Just like neurons, nodes of a layer are connected to the nodes in the previous and the following layers. These connections in the computational graph are called edges. In an ANN, the edges are weighted and usually just referred to as weights.

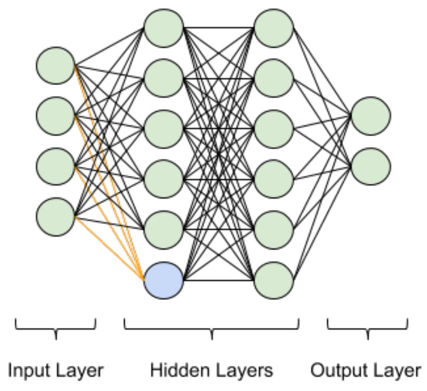


Figure 2.5: A visualization of a “deep” artificial neural network. The circles are nodes, and the connection lines are weights. The blue node value is calculated by using the orange colored weights and the connected nodes.

Given an input, the network moves forward by calculating the values of the nodes layer-by-layer. The node values of the input layer are equal to the given input. At each following node, the value is calculated by having the weighted sum of the node values in the previous layer. For example, in Figure 2.5, the value of the blue node is calculated by multiplying each previous node with the corresponding weight and summing them all together. Equation 2.6 shows the node value calculation where n is the number of nodes in the previous layer, x_i is a previous node value, and w_i is a weight. After each node is calculated in the network, the node values of the output layer give us the prediction. This process is called ”inference”.

$$z = \sum_{i=1}^{n} x_i \cdot w_i \tag{2.6}$$

To train a neural network, first, the weights are initialized randomly. The network immediately starts predicting by using these random weights, as it was already trained. These predictions are compared to the given correct labels to calculate an error rate. A loss function does the error calculation. This is followed by the backpropagation process, where the network determines how much each weight contributes to making the errors. Then, the weights are adjusted in a way to minimize errors in the next run. The weights that contribute more to the errors are changed more. Therefore, the weights are adjusted to reduce the loss. After the adjustments, the neural network moves forward to make new predictions. This process is iterated, and at each iteration, the loss is minimized.

The idea of artificial neural networks has been around since the invention of computers. However, they started to outperform the other machine learning methods in the last ten years. Their true potential discovered first on processing the images (such as [21]). However, similar techniques started to demonstrate the same success in text processing, as well.

Hereunder, we explain the main components of a neural network in detail, including activation, loss, weight optimization, and regularization.

2.3.1 Non-linear Activation Functions

In the previous section, we explained how the node values are calculated during the inference process. In Eq. 2.6, we display a linear function as a starting point. If we only use linear activation functions, the output layer will be a linear function of the input layer. In such a case, regardless of how many hidden layers we use, the neural network will act as a single layer. Therefore, to escape the linearity, non-linear activation functions are introduced. The activation functions take the linear weight sum as input and output a modified version. Although there are many activation functions (such as *sigmoid* or *tanh*), the general equation can be written as shown in Eq. 2.7 given that σ is the activation function.

$$z = \sigma(\sum_{i=1}^{n} x_i \cdot w_i) \tag{2.7}$$

One commonly used activation function is the Rectified Linear Unit (ReLU) [27]. ReLU is a simple yet effective activation function. If the input is greater than zero, ReLU returns the original input. For any input smaller than 0, ReLU returns 0. It can be defined as the following:

$$\sigma_{relu} = max(0, x) \tag{2.8}$$

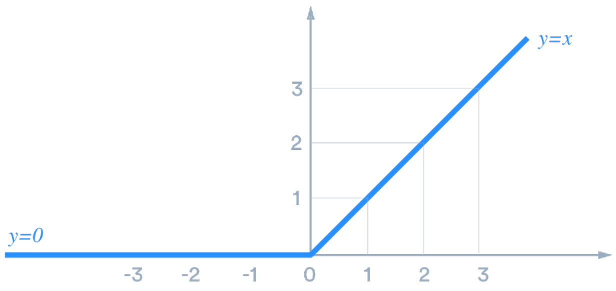


Figure 2.6: Plot of the ReLU function.

Thus, this makes ReLU, partially linear. It acts linearly for the values larger than zero and non-linear for the values smaller than zero (see Figure 2.6). The linearity makes the training faster compared to other functions such as the exponential ones. Setting the values with negative inputs to zero leads to sparsity [11]. Besides speeding up the processing, the sparsity also produces models with less noise since only specific nodes are activated.

2.3.2 Loss

As also mentioned in Section 2.3, a loss value is calculated at the end of each iteration. The idea of training a neural network is searching for the ideal configuration of the weights. The loss represents the rate of the errors done by a neural network with the current weights. When two configurations are compared, the one generates the smaller loss is accepted as better, because it makes fewer errors.

Various loss functions are useful on different types of neural networks based on the task and the data at hand. Cross-entropy [12] is one of the most common loss functions that is used for classification tasks. In a binary classification setup, there are two possible outcomes to be predicted. Considering the classes A and B; the log probabilities of each instance being A and B are calculated. Then, the sum of the log probabilities across the dataset is divided by the total number of instances. Given that y_A and y_B are predictions, $P(y_A)$ and $P(y_B)$ are the probabilities that the instances being A or B, and N is the number of points in the dataset, Equation 2.9 defines the cross-entropy loss function.

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_{A,i}log(P(y_{A,i})) \cdot y_{B,i}log(P(y_{B,i}))) \tag{2.9}$$

2.3.3 Weight Optimization

Neural networks are always in search of the optimal weight configuration that gives the minimum loss. Optimization algorithms (or optimizers) are responsible for updating the weights to accomplish this. The weight updates are calculated based on the initial weight, the loss, and the learning rate. The learning rate is a hyperparameter that gives us some control over how much the neural network is adjusting the weights.

Hereunder, we describe the most common three weight optimization methods; Gradient Descent, Root Mean Square Propagation, and Adaptive Moment Estimation.

Gradient Descent

The most basic optimizer, Gradient Descent [38] is the starting idea of many other optimization algorithms. We, first, define the gradient of the loss function L with respect to the weight w as:

$$g = \bigtriangledown_w L(w) \tag{2.10}$$

Gradient Descent updates a weight w, by subtracting the gradient g multiplied by the learning rate η.

$$w \leftarrow w - \eta \cdot g \tag{2.11}$$

Root Mean Square Propagation

A commonly used optimization algorithm is the Root Mean Square Propagation (RMSprop) [45]. It adjusts the learning rate for each weight by using the gradient of the previous updates. At each time step t, the exponential average of the past squared gradient is calculated until the last update. Then it is added to the square of the current gradient (see Eq. 2.12). This technique gives us the exponential average of the squared gradient calculated at the current step t.

Here we also introduce another hyperparameter represented as γ. We multiply the exponential average at the time $t-1$ with γ, and the square of the current gradient with $(1-\gamma)$. This approach allows weighing the recent gradients more than the previous ones. Thus the equation of exponential average of the square of the gradient becomes:

$$E[g^2]_t = \gamma \cdot E[g^2]_{t-1} + (1-\gamma) \cdot g_t^2 \tag{2.12}$$

RMSprop adjusts the learning rate by dividing it into the square root of the exponential average of the current gradient. A constant ϵ is also added to the exponential average to avoid a division by zero. When we place this adjusted learning rate calculation into the Gradient Descent update function, we obtain the RMSprop update function (Eq. 2.13).

$$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{2.13}$$

Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [18] is another optimization algorithm that computes an adaptive learning rate for each weight. Similar to RMSprop, Adam calculates the exponential average of the past squared gradients v_t as:

$$v_t = \beta_1 \cdot v_{t-1} + (1 - \beta_1) \cdot g_t^2 \tag{2.14}$$

As an addition, Adam also calculates the exponential average of the past gradients m_t as:

$$m_t = \beta_2 \cdot m_{t-1} + (1 - \beta_2) \cdot g_t \tag{2.15}$$

Both of the v_t and m_t calculations have the hyperparameters, β_1 and β_2, that works similar to the hyperparameter γ used in RMSprop. The v_t and m_t are initialized as vectors of zero. It is observed that they are biased towards zero [18]. This bias is corrected by:

$$v'_t = \frac{v_t}{1 - \beta_1} \tag{2.16}$$

$$m'_t = \frac{m_t}{1 - \beta_2} \tag{2.17}$$

Finally, the Gradient Descent update function is modified with using bias-corrected v'_t and m'_t to obtain the Adam update function calculated as:

$$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{v'_t} + E} \cdot m'_t \tag{2.18}$$

2.3.4 Regularization

The weights of a neural network are optimized during the training operation. It is common to observe that the weights are converged only for the dataset used for training the neural network, meaning that the model overfits. Regularization methods are developed to prevent neural networks from overfitting their weights on the training dataset.

Regularizers add a regularization term $R(w)$, to the loss calculation. The regularization term is chosen to penalize the complexity of the neural network. The importance of the regularization is controlled by adding a hyperparameter λ. Therefore, the loss is increased by the regularization term on the weight and λ. Given that $L(w)$ is the loss with respect to the weight w, the loss calculation is updated as in equation 2.19.

$$\hat{L}(w) = L(w) + \lambda R(w) \tag{2.19}$$

Regularizers are forcing the weights towards zero. This way, the values in the weight matrix are reduced. The input values of the activation function become smaller. Thus, the effect of the activation function is lessened. The complexity of the neural network is simplified, which in turn reduces the overfitting [22].

There are two commonly used regularization terms, L1 and L2. L1 (or lasso) regularization is calculated as shown in Equation 2.20. L1 regularization achieves the complexity reduction by moving weights to zero and creating sparsity in the weight matrix.

$$R_{L1} = \sum_{i=0}^{n} |w| \tag{2.20}$$

L2 regularization (or weight decay) penalizes the network by the sum of the squares of all the weights (see Eq. 2.21). Thus, L2 regularization never makes the weights zero and provides a non-sparse solution.

$$R_{L2} = \sum_{i=0}^{n} |w^2| \tag{2.21}$$

Here in this study, we empirically decided to use L2 regularization with $\lambda = 0.001$ in the fully connected layers in the neural networks.

2.4 Text Representations

The mathematics behind machine learning and neural networks is based on processing numerical values in vectors or matrices. It is crucial to have a method that transforms text into a numerical representation. This method allows computers to process unstructured text. Effective feature extraction is the key to robust and high performing automated classification systems. Researchers have suggested different feature extraction techniques to encode text. Hereunder, we mention both classical and recent encoding methods that are used in feature extraction for unstructured text.

2.4.1 Bag-of-Words

Bag-of-Words (BoW) representation method is one of the most widely used ways to represent the text as numerical vectors [52]. The idea is structuring a vocabulary of the unique words that exist in the corpus. Then, each text (such as a sentence) is encoded as a vector with the length of this vocabulary. In this vector, each word carries a weight. At its most basic form, this weight can be a binary value which indicates the presence or absence of the word. Another weight calculation can be using the term frequency that counts the number of times a word appears in the text.

When the term frequencies are used as weights, some frequently used words dominate the representations that might not provide distinctive information. To avoid this domination, Term Frequency - Inverse Document Frequency (TF-IDF) approach has been proposed [43]. TF-IDF calculation multiplies the term frequency with the log inverse document frequency of that word (Eq. 2.22). This penalizes the words that are frequent in the corpus. With this score, it is possible to give larger weights to the words that are distinct in the text.

$$w_{i,j} = tf_{i,j} \cdot log(\frac{N}{df_i}) \tag{2.22}$$

The words with too high or too low document frequencies can be seen as outliers. The words with too high document frequency occur in almost all of the documents. Using them to encode a document does not help to differentiate between the instances. This makes them unimportant for the classification tasks. Likewise, the words with too low document frequency are almost unique to the given text. Using them in the encodings may create an unwanted bias that shifts towards those words instead of the collective meaning. Also, in practice, reducing the dimensionality of the vectors is beneficial to reduce the computational complexity. Thus, the maximum document frequency parameter is set to prevent adding too frequent words to the BoW vocabulary used to encode the documents. The minimum document frequency is set to prevent adding the words that do not frequently occur in different documents to the vocabulary.

Although BoW+TF-IDF methods show promising results, they disregard the word order, context and grammar. One solution for including word order and context is to create a BoW structure with n-grams words instead of taking each word separately. With n-gram representations, BoW can capture more information compared to unigram approaches. Although word order and context are preserved, syntactic representations are unable to capture the word meanings, and so, cannot encode word similarity. Another fundamental problem with BoW is the sparsity. BoW representations yield high-dimensional sparse vectors. Thus, there is a pressure for vocabulary decrease.

2.4.2 Word Embeddings

Word embeddings are dense high-dimensional vectors of words, capable of capturing semantics. There have been various methods proposed to generate word embeddings over the past years. They are all generated by neural networks trained over a very large corpus. The recent methods managed to encode the relationships in text and semantic similarities between words, unlike BoW approaches.

Word2Vec & GloVe

One of the most well-known approaches is word2vec embeddings, which has used skip-grams and continuous bag-of-words (CBOW) to encode each word [25]. CBOW and skip-gram methods calculate the relations between a word and its context by measuring the distances between the word itself and its surrounding words. In this method, a shallow neural network is trained to perform a language modeling task. The neural network encodes the words together with their given local contexts. After the training, by taking the weight matrix of the hidden layer in this shallow network, we obtain the vector representations of the words. Finally, these representations can be used to compute the similarities (or relations) between the words.

A similar approach is employed by the model known as GloVe embeddings. Word2vec and GloVe approaches have the same mathematical base other than minor hyperparameters [31]. A major difference is that GloVe builds a global co-occurrence matrix of the words.

Embeddings from Language Models

Building on the idea of aforementioned models, a *contextualized word embedding* model named Embeddings from Language Models (ELMo) recently developed and showed promising performance on benchmarks [32]. Contextualized word embeddings also follow the idea of training a neural network language model approach. As a difference, they do it with complex deep neural networks that employed bidirectional language modeling (biLM).

ELMo is trained on a prediction task, where the model encodes sequences while learning to predict the next word. At the training of the ELMo model, for a given input sequence, first, a vector representation for each word is obtained by Convolutional Neural Networks (CNN) layers. Then, the sequence of these vectors is passed to a group of forward Long Short-Term Memory (LSTM) layers. Each LSTM layer yields a contextual representation of the words. The output of the top layer LSTM is given to a softmax layer to have a prediction on the next word. Another group of LSTM layers encodes the sequence in the same way but in the reverse order. Thus the bidirectional language model is a concatenation of these individual forward and backward LSTM layers that jointly maximizes the log-likelihood. Given that, θ_x is the initial word representation, the θ_s is the output of the softmax layer, and N is the number of words in the sequence, the formulation of the biLM is:

$$\sum_{k=1}^{N}(\log p(t_k|t_1,\ldots,t_{k-1};\theta_x,\vec{\theta}_{LSTM},\theta_s) + (\log p(t_k|t_{k+1},\ldots,t_N;\theta_x,\overleftarrow{\theta}_{LSTM},\theta_s) \tag{2.23}$$

The goal of a trained word2vec and GloVe models is to create a universal word representations matrix that carries the most frequent meaning of the words. Thus we can look for the representation of a word from that matrix. As a major difference, once a contextualized word embedding model is trained, it generates representations over the entire sequences instead of individual words. contextualized word embeddings manage to produce representations that implicitly carry the meaning of the word in that particular context. For example, a contextualized word embedding would return different vector representations for the mentions of the word *"bank"* in *"bank account"* and *"bank of the river"*[1]. Hence, it prevents confusion with potentially ambiguous words.

Although the ELMo model is not considered fine-tunable as much as a transfer learning approach, it can be highly optimized to different downstream NLP tasks and different domains. This is done by collapsing all layers into a single vector and weighing the bidirectional language model based on the task. Given that γ^{task} is a weight that allows the downstream NLP model to optimize the ELMo vector and s^{task} is a softmax-normalized vector, the formulation of the optimization is:

$$ELMO_k^{task} = \gamma^{task}\sum_{j}^{L} s_j^{task} h_{k,j}^{LM} \tag{2.24}$$

Bidirectional Encoder Representations from Transformers

Another method that recently set the state-of-the-art in the word encoding field is another contextualized word embedding model named Bidirectional Encoder Representations from Transformers (BERT) [10]. BERT is built upon the other contextualized word embedding models, including ELMo and has the same goals and behaviours.

BERT introduces a model that applies the bidirectional encoding at a level as deep as the hidden layers. Bidirectionally training a language model on the next word prediction task is not trivial. In the bidirectional setup, a word is predicted by reading all of the surrounding words from left to right and right to left. The language model already knows about the entire sequence when we attempt to predict the next word. Consequently, the model knows the answer without "learning" anything. This problem is known as the word "seeing itself" in the context of another word. BERT approach overcomes this

[1]Example is taken from https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html. Accessed on 9 January 2020

issue by using masked language modeling (MLM). In MLM technique, 15% of the words in a sequence is randomly replaced with a placeholder token, [MASK], and only these placeholders are predicted. Thus, during the training, BERT model does not “see” all the words at once. At the final step, the vectors from the final hidden layer are given to an output layer with softmax activation.

On top of next word prediction training, BERT is also trained on next sentence prediction to learn the relationship between sentences. During the training, BERT model is given pairs of sentences labeled as ‘IsNext’ and ‘NotNext’. Thus the BERT encoders are trained beyond the local context and also encodes information from surrounding sentences.

Finally, BERT is designed to be a transfer learning model. Thus, all the parameters of pre-trained BERT models can be fine-tuned end-to-end to model a downstream NLP task.

Pre-trained Language Models

To truly benefit from the power of the neural word embedding algorithms, the model needs to be trained on a large collection of textual data. Even though the word embedding algorithms are unsupervised learning techniques, processing such amounts of data is time taking and requires powerful hardware. As this is a commonly known issue in the research area, it is possible to find word embedding models shared publicly often by the developers of the algorithms. Such shared models are trained on a large set of textual data from multiple domains, and so, they aim to provide a general encoding of natural language text regardless of the downstream NLP task or the specific domain.

Chapter 3

Approaches

In this chapter, we define two methodologies that perform sentence classification. In the first part, we introduce the context-agnostic approach, and in the second part, we describe the context-dependent approach. Likewise, we explain how do we configure specific algorithms to fit our purposes.

3.1 Context-agnostic Event Classification

In a news article, many sentences form a single integrated story. Since our task is classifying sentences, we define the context of a sentence as the other sentences surrounding it. In this section, we follow the context-agnostic approach. The context-agnostic approach aims to classify a given input sentence by only using its properties. Thus, the other context surrounding the input is not taken into account. In other words, we process each sentence independent from its context. Our general propose approach to sentence classification is based on artificial neural networks. Thus, in this section, we explore creating neural network models using only the words or characters of the sentence.

Figure 3.1 illustrates an overview of the neural network architecture we propose for context-agnostic approach. As can be seen in the figure, the input of the network is a single sentence. Then, the sentence is given to a word embedding model as a whole to obtain the word encodings for each word used in its content. We use two different word embedding models, namely ELMo and BERT, implemented in separate neural networks. In the figure, the graph on the left shows a network with ELMo word embeddings and the other shows a network with BERT word embeddings. The two embedding models are integrated with different strategies which are explained in Section 3.1.1.

The word embedding layers return a vector for each word in the content of the sentence. These vectors are given to a pooling layer to reduce the dimensions before using the classification models, which is explained in Section 3.1.2. Thus, we obtain a single vector representation for each input sentence. Finally, the sentence encoding is fed into the classification layers (red area in Figure 3.1) to decide on the label of the sentence. We define multiple architectures as the classification layers. However, for demonstration purposes, we visualize our most straightforward context-agnostic architecture (referred to as Model A) in the figure. All of the architectures are introduced in Section 3.1.3. The different network architectures are obtained by implementing different classification layers. Thus, the rest of the networks stays the same for any context-agnostic model.

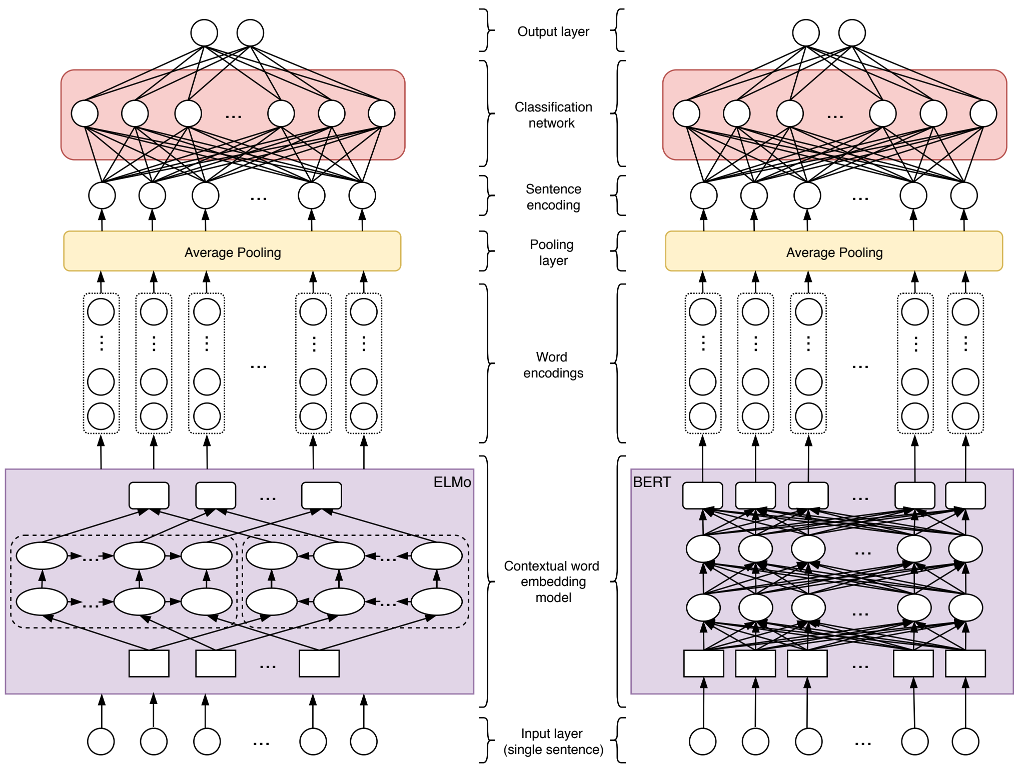


Figure 3.1: The Model A neural network architecture in the context-agnostic classification approach.

3.1.1 Word Embedding Model Integration

The sentence encoding aims to generate numerical vector representations of a given sentence. Generating sentence representations can be done in many ways, as discussed in Section 2.4. A sentence is a set of words ordered in a meaningful way and can be encoded by using the properties of the words that it contains. Following this idea, our approach to get sentence representations starts with encoding each word in a given sentence using contextual word embeddings. Due to the data and hardware constraints, instead of training a new model, we make use of the general-purpose word embedding models that are trained on a large scale corpus. More specifically, we incorporate two state-of-the-art encoding algorithms that have publicly available pre-trained models, namely ELMo and BERT.

There are currently two strategies to incorporate pre-trained language models to downstream classification tasks; feature-based and fine-tuning. In the feature-based strategy, the pre-trained model is utilized as an external system. We send a word and its context as an input to the pre-trained model. Then, the model returns the word embedding calculated for that word. The weights of the embedding model do not change based on the downstream task during training. We integrate the ELMo embedding model with feature-based strategy. For this, we use the pre-trained model of ELMo trained on 1 Billion Word Benchmark [7] dataset. The model is hosted by Google on TensorFlow Hub[1].

[1]Models hosted by Google on TensorFlow Hub: https://tfhub.dev/google/. Accessed on 9 January 2020

In the fine-tuning strategy, we consider the pre-trained word embedding model as a group of trainable hidden layers and embed them into our neural network architecture. During the training of the entire neural network, the weights of the word embedding layers are also reconfigured. Thus, the parameters of the model are fine-tuned on our domain-specific text. All the parameters of pre-trained BERT models can be fine-tuned end-to-end to model a downstream NLP task as it is designed to be a transfer learning model. Thus, we integrate the pre-trained BERT model with the fine-tuning strategy. We use the uncased BERT model with 12 hidden layers, trained on Wikipedia and BookCorpus data and hosted by Google on TensorFlow Hub. Although the recommended way is fine-tuning all the 110,000,000 pre-trained parameters, we could only fine-tune the last four layers (28,000,000 parameters) of the model with our hardware setup that consists of a single NVIDIA GeForce 1080 Ti graphics processing unit with 11 gigabytes of memory.

3.1.2 Word to Sentence Encodings

By default, the pre-trained word embedding models are trained to return a vector per given word (Figure 3.1). For a sentence s is $\{w_1, w_2, w_3, \ldots, w_L\}$, w represents a word and L is the number of words. If the pre-trained word embedding model returns vectors with length D per w, the sentence s is represented by a matrix of size $L \times D$. Given that for ELMo model $D = 1024$ and for BERT model $D = 768$, training our neural network on $L \times D$ is computationally expensive. Thus, the final step of our encoding approach consists of applying a function to reduce the encoding dimensionality from $L \times D$ to a vector with length D, as shown in Figure 3.2.

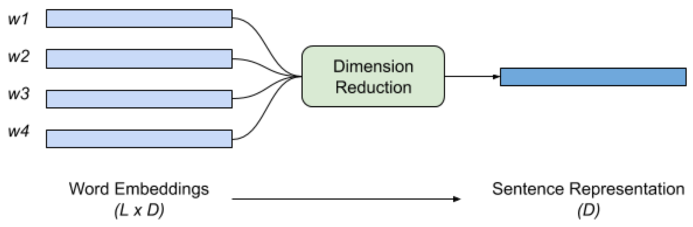


Figure 3.2: Average pooling strategy is used to reduce the dimensionality of the representations from $L \times D$ matrix to a vector with length D.

As the dimensionality reduction strategy, we apply the average pooling method. In average pooling, we take each information into account and transfer it to the next layer, unlike max-pooling where only the largest of the features are selected. We apply the average pooling linearly. For word embedding vectors with features $\{f_1, f_2, f_3, \ldots, f_D\}$, we calculate the sum of each n^{th} feature f_n from each vector and then divide it by the total number of words, L. The result is the nth feature of the sentence embedding vector $s^{'}$ (Eq. 3.1).

$$s^{'} = \{\frac{\sum_{i=1}^{L} f_{1_i}}{L}, \frac{\sum_{i=1}^{L} f_{2_i}}{L}, \ldots, \frac{\sum_{i=1}^{L} f_{D_i}}{L}\} \tag{3.1}$$

As an alternative to the average pooling method, adding an additional Long Short-Term Memory (LSTM) layer is also an effective way to obtain sentence representations. However, using LSTM layers increases the computational complexity, and our preliminary experiments showed that it does not significantly improve over the pooling method.

3.1.3 Label Predictions

After the encoding, the sentence representations are passed to the next layers that further process them to decide on the labels. The rest of the architecture is defined by us. We experiment with different network architectures to demonstrate the full potential of the neural networks. The architectures are created by varying the types of the hidden layers, adding residual connections, the order of the layers, using dropout, and using batch normalization. The number of hidden units, kernel sizes and dropout rates for each of the networks are heuristically decided based on the output sizes of the layers.

All of the given architectures are run once with integrating the ELMo embedding model and once with integrating the BERT embedding model. Below the 'encoding network' refers to ELMo and BERT models (explained in Section 3.1.1) and the 'pooling layer' refers to the average pooling method (explained in Section 3.1.2). In the rest of this thesis, we refer to different architectures by their index (e.g., ELMO-based Model A).

- **A.** The encoding network and pooling layer are followed by a single fully connected (FC) layer with 256 hidden units (Appendix, Figure A.1).
- **A2.** More sophisticated version of the above Model A; the encoding network and pooling layer are followed by a FC layer with 256 hidden units, batch normalization, dropout with 0.5 rate, another FC layer with 128 hidden units, and a batch normalization (Appendix, Figure A.2).
- **B.** The encoding network and pooling layer are followed by 4 FC layers with 256 hidden units. For a network with hidden layers $\{l_1, l_2, l_3, l_4\}$, a residual connection is implemented in a way that adds the outputs of l_1 to the outputs of l_3 before continuing to l_4 (Appendix, Figure A.3). To perform the addition, we use the 'Add' layer[2] provided by the Keras Python package.
- **B2.** A sophisticated version of Model B. The only difference is the batch normalization and dropout with 0.3 rate applied after each FC layer within the residual block (Appendix, Figure A.4).
- **C.** The encoding network and pooling layer are followed by two sets of 1-dimensional (1-D) convolutional layer with 64 filters and 5 kernel size followed by a 1-D max-pooling layer. Then the output is flattened and fed into a FC layer with 512 hidden units (Appendix, Figure A.5).
- **D.** The encoding network and pooling layer are followed by two 1-dimensional (1-D) convolutional layer with 64 filters and 5 kernel size. However, the output of the first convolutional layer is added to the second convolutional layer by a residual connection (in the same way with Model B). After the addition of the outputs, a 1-dimensional max-pooling layer is applied, the output is flattened and fed into a FC layer with 256 hidden units (Appendix, Figure A.6).

For any of the above neural models, the output layer is a fully connected layer with 2 units and softmax activation. The number of output units is due to using the one-hot encoded versions of our labels. In one-hot encoding, our positive class is represented as $[0, 1]$ and the negative class is represented as $[1, 0]$.

The implementation of the neural networks is done by using Keras version 2.2.4 [8] and Tensorflow version 1.14 [1] Python libraries.

[2] Add layer: https://keras.io/layers/merge/#add. Accessed on 9 January 2020

3.1.4 Other Network Configurations

Besides the network architecture, we determine the other configurations of the network, including the loss function, number of epochs, the optimizer function, and the learning rate:

- **Loss function:** Our task at hand is essentially a binary classification problem. Thus, we use the binary cross entropy method as the loss function for all of the networks for the context-agnostic approach.
- **Loss optimizers:** We experiment with two of the most common optimization algorithms: Adam and RMSprop.
- **Epochs:**
  - Networks with the ELMo model: The number of epochs is heuristically set to 50. We also save the model each time an epoch is finished. In the end, we select the best performing model to be used in the test.
  - Networks with the BERT model: The developers of the BERT model suggests fine-tuning it for one or two epochs, and thus, we run BERT for one epoch.
- **Learning Rate Reduction:** We use a method to reduce the learning rate proportionally during the training of the algorithm if the performance score does not improve after a number of epochs. For our experiments, we have set it to reduce the learning rate by a factor of 0.1 if no improvement is observed after 3 epochs. However, we also set a minimum learning rate as 0.00001 to prevent too small learning rates which would potentially prevent the learning. This is only applied in ELMo-based networks since BERT-based networks are only trained for one epoch.
- **Learning rates:** For both ELMo and BERT versions of our networks, we experiment with multiple learning rate values. However, these values are set differently for ELMo and BERT-based versions.

3.1.5 Baselines

To have a better understanding of the performance of our neural network-based approach, we compare it against a baseline based on classical machine learning algorithms. More specifically, we train Decision Trees, Random Forests, and Support Vector Machines classifiers on the Bag-of-Words encodings using TF-IDF method to calculate the weights for the encodings. Because our task is classifying the sentences, the document frequency for us is the total number of occurrences of words in unique sentences (see Section 2.4.1). Moreover, we limit the vocabulary by setting upper and lower boundaries on the document frequency, instead of using each token present in the corpus. These limitations are listed below:

- *Maximum document frequency* was set empirically by trying different values and observing the results on a validation set. For this study, the optimal ratio is set to 12%. This ratio helps us to exclude words generally accepted as stop words, such as "of", "that", "and", as much as the corpus specific ones.
- *Minimum document frequency* was set empirically as well. We set this condition to capture the words that occur at least in 10 different documents. Thus, any word used in less than 10 documents is not included in the vocabulary.

- We empirically discovered that the numerals also do not contribute to distinguishing instances with different classes in this task. Thus numerical values between 0 and 10.000 are not included in the vocabulary.

In addition to the classical machine learning algorithms, we also experiment with training a neural network by using the same TF-IDF vectors in order to understand the contribution of using contextual word embeddings. More specifically, we use the Model A explained in Section 3.1.3, with TF-IDF vectors as input.

The baseline models are trained and tested on the same dataset as the neural network approach in order to be comparable to each other. The hyperparameters of the baseline algorithms are optimized by the grid-search method. Each parameter combination is used to train a model on the training set. Then, each of these models is evaluated on the validation set, and the performances are recorded. The model with the best performing hyperparameter combination is selected to be evaluated on the test set. Finally, the test set results are compared to the proposed neural network-based models.

The best performing hyperparameter combinations per algorithm are listed below.

- Decision Tree algorithm: min samples split: 170, max features: None, max depth: None, min samples leaf: 5.
- Random Forest algorithm: n estimators: 300, max depth: None, max features: 0.2, bootstrap: True, min samples split: 10.
- Support Vector Machine algorithm: kernel: 'linear', C: 3.

The implementation of the classical machine learning algorithms is done by using the 0.20.0 version of the Scikit-learn [30] Python library.

3.2 Context-dependent Event Classification

In the previous section, we considered each sentence as an independent instance. Each sentence is deliberately written in relation to its preceding and following sentences to create a coherent document entity. To put it another way, the consecutive sentences in an article tell a single story collectively. If they are considered separately, they might represent different meanings and events. As humans, we also process the information in an article as a storyline and consider all previous and next information during understanding. In this section, we follow a context-dependent approach. We modify our network architecture to detect event-related sentences by taking their preceding and following sentences into account.

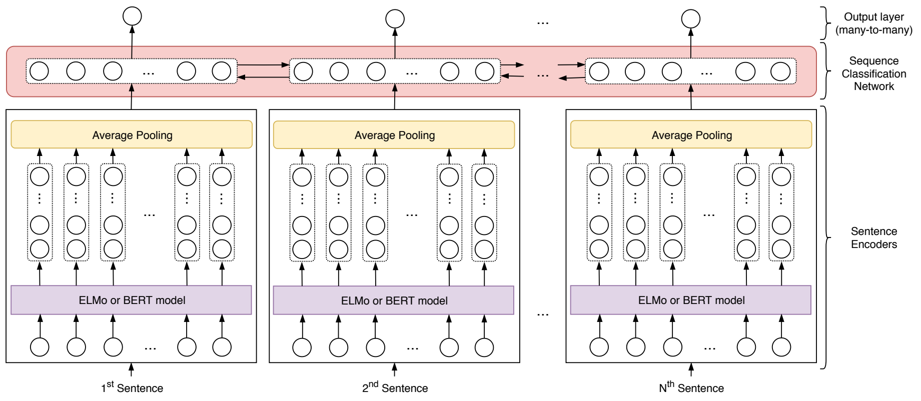


Figure 3.3: The general neural network architecture in the context-dependent classification approach. Here the figure demonstrates a network for article-as-sequence strategy. For the sliding window strategy, only the output layer would be different (see Section 3.2.1).

Our approach can be defined as a sequence classification of the sentences. In sequence classification, the inputs are given as a sequence over space and predictions are made for the entire sequence. As can be seen in Figure 3.3, multiple sentences are given to the neural network at once. For context-dependent sentence classification, we propose two different strategies which are explained in Section 3.2.1. Initially, each sentence is encoded independently from the others and encoded only by the words that it contains. The encoding is done by the aforementioned word embedding models (see Section 3.1.1) and the average pooling method (see Section 3.1.2). Then the encoded sentences are fed into the sequence classification layers of the network (red area in Figure 3.3). In the figure, the classification part consists of a single Long Short-Term Memory layer for visualization purposes. However, we define multiple and more complex architectures which are introduced in Section 3.2.2. The different architectures are obtained by varying the classification layers. Thus, the rest of the networks would stay the same for any context-dependent model mentioned in Section 3.2.2.

3.2.1 Sequencing Strategies

Sliding window strategy

We create sequences of sentences by a sliding window. Each sentence is turned into a sequence consists of itself and its surrounding sentences. And, each sequence contains the preceding sentence(s), the corresponding sentence, and the following sentence(s). The number of preceding and following sentences in the sequence is controlled by a hyperparameter called window-width. Thus the sliding window strategy allows us to control the context that should be taken into account to classify the corresponding sentence.

If we create a sequence with window-width set to 3 for a sentence at position i, the sequence will be $\{sentence_{i-1}, sentence_i, sentence_{i+1}\}$, as exemplified in Table 3.1. Naturally, the first sentence in an article does not have a preceding sentence, and the last sentence does not have a following one. For the situations with a missing preceding and/or following sentence, we place a padding sentence that contains a single word saying "PADDING". In our experiments, we run each of the sequence classification networks with window-width set to 3, 5, and 7.

Corresponding Sentence	Generated Sentence Sequence
S_1	$S_{padding}, S_1, S_2$
S_2	S_1, S_2, S_3
S_3	S_2, S_3, S_4
S_4	$S_3, S_4, S_{padding}$

Table 3.1: Example sequence generation. Here the example article consists of 4 sentences in total and the window-width is set to 3. Notice how we use the padding sentence for the non-existent sentences.

The sequences are fed into the sequence classification part of the network that consists of LSTM layers. The sliding window strategy aims to decide on the label for the corresponding sentence by encoding the entire given sequence. Thus, the output layer is configured to return only one prediction, even though the input contains many sentences. This is also known as many-to-one architecture. Figure 3.4 shows the difference in the output layer compared to aforementioned architecture in Figure 3.3

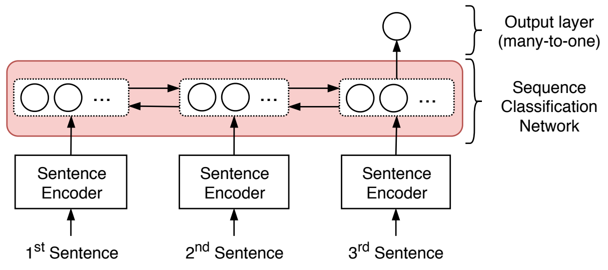


Figure 3.4: Simplified visualization of the context-dependent neural network with the many-to-one output layer. As an example, we show the network with window-width hyperparameter set to 3.

Article-as-sequence strategy

In an article, every sentence interacts with others in order to create a single meaningful unified piece. Moreover, some of the sentences contain distant relations; the same information can occur at the beginning of an article and again at the end of it. Here, we use the article-as-sequence strategy to capture such distant relations and classify each sentence using the information in the entire article.

Sentences in an article are written one after another, making each article a natural sequence of sentences. This allows us to create the sequences from all of the sentences of the article. To be able to process the sequences in the rest of the network, they need to be at the same length. The articles, however, contain a varying number of sentences. We define another hyperparameter, sequence-length, and adjust the article sequences to be at that length. If an article contains fewer sentences than the number in sequence-length parameter, we add padding sentences to the end of the article. For instance, if sequence-length parameter is set to 6 and the article contains 3 sentences, the sequence will be $\{sentence_1, sentence_2, sentence_3, sentence_{padding}, sentence_{padding}, sentence_{padding}\}$.

For the ELMo-based networks, we use a padding sentence that contains a single word "PADDING" that is sent to ELMo model to be encoded. In the BERT-based networks, we feed BERT model with arrays contain only "0" (zero) as the padding.

The downside of this strategy is that the final sentences of an article are pruned if the article contains more sentences than the sequence-length. For our study, we set the sequence-length to the maximum article length found in our dataset used in our experiments (detailed in Section 4.1.1). Thus, in our study, we have encoded and analysed every sentence in each article, leaving none of them out.

3.2.2 Label Predictions

In the next step, these sequences of sentence representations are passed to the layers that do the sequence classification to decide on the labels. We focus on using a special neural network architecture that can process sequences of inputs, namely Recurrent Neural Networks (RNN). Traditional RNNs suffer from vanishing gradient problem where the update on the weights of the earlier layers becomes vanishingly small with each back-propogation. As a result, the earlier weights do not contribute to reducing the loss; in other words, the connection with them is lost.

We use Long Short-Term Memory (LSTM) [14] layers that are capable of handling the vanishing gradient problem and able to build long-term dependencies. At each time step LSTM calculates whether the information should be remembered or forgotten through its forget gate weights. The forget gate activations are close to 1, which prevents gradient from being multiplied by smaller values and get vanished over time.

LSTM implementation follows a unidirectional method reading the data from the beginning to the end (or vice versa) and construct the dependencies. It only considers information that has seen by it so far. Here, we use the Bidirectional LSTM (BiLSTM) approach [16], which reads the data once from the beginning to the end and once from the end to the beginning. Thus the information for each data point is encoded with both preceding and following information taken into account.

In order to create an optimal architecture for classification, we run experiments with different versions of our network. The network architectures are created by varying a fully connected layer, adding residual connections, the order of the layers, and using dropout. The number of hidden units and dropout rates for each of the networks are heuristically decided based on the output sizes of the layers.

All of the given architectures are run once with integrating the ELMo embedding model and once with integrating the BERT embedding model. Below the 'encoding network' refers to ELMo and BERT models explained in Section 3.1.1 and the 'pooling layer' refers to the average pooling method explained in Section 3.1.2. In the rest of this thesis, we refer to different types of architectures by their index (e.g., ELMO-based Model E). The naming starts from 'E' with the aforementioned network architectures to prevent the confusion.

- **E.** The encoding network and pooling layer are followed by a single BiLSTM layer with 256 hidden units and a dropout regularization with 0.4 rate (Appendix, Figure A.7).
- **F.** The encoding network and pooling layer are followed by two BiLSTM layers with 256 hidden units. The output of the first BiLSTM layer is added to the second BiLSTM layer to form a residual connection. After the addition of the outputs, a dropout with 0.4 rate is applied (Appendix, Figure A.8).
- **G.** The encoding network and pooling layer are followed by a FC layer with 256 hidden units and a BiLSTM layer with 128 hidden units (Appendix, Figure A.9).
- **H.** The encoding network and pooling layer are followed by a FC layer with 256 hidden units and two BiLSTM layers with 256 hidden units. The output of the first BiLSTM layer is added to the output of the second BiLSTM layer to form a residual connection before the output layer (Appendix, Figure A.10).

The output layers of all ELMo-based networks are an LSTM layer with two units and softmax activation. Our preliminary experiments on the validation set have shown that the context-dependent BERT-based networks perform better with sigmoid activation than the softmax activation. Thus, we used sigmoid activation for BERT-based networks.

The same neural network configurations explained in Section 3.1.4 are also applied to the context-dependent neural networks. Additionally, we accept the context-agnostic neural network models as our baselines that help us to assess the contribution of the context-dependent approach.

The neural network models are implemented by using Keras version 2.2.4 [8] and Tensorflow version 1.14 [1] Python libraries.

Chapter 4

Evaluation

In this chapter, first, we describe the dataset and give a statistical analysis of it. Next, we elaborate on the evaluation metric we used. Finally, we present the results of our experiments.

4.1 Experimental Setup

4.1.1 Dataset

We use ProtestNews dataset for evaluating our experiments. The dataset consists of 587 English written Indian news articles, published in India between 2001 and 2017. The news articles were collected, scraped, and annotated by the Department of Sociology in Koç University, in the scope of the ERC funded Emerging Welfare project [17].

In the first step of the annotation process, the news articles were annotated at the document level with "event" or "non-event" labels[1]. Scope of the project is narrowed to find protest events that were happened or happening at the time of the article published. Hence, the annotators only labeled an article as an "event", if the article mentions past or ongoing protest events. In the second step of the annotation process, 8,337 sentences were manually annotated as "event" and "non-event". Both articles and sentences were assessed by two annotators, and the disagreements were adjudicated by a third person. The annotators were graduate students in social and political sciences.

To mitigate working with imbalanced data, we consider only the articles that are manually annotated as protest-related. Some articles do not contain any sentences labeled as "event", meaning that all the sentences in those articles were annotated as "non-event". This causes an imbalance between the negative and positive class. To maintain the balance, we dropped such articles without any "event" labeled sentences from our dataset. As a result of this elimination process, there were 315 articles left in total. We refer these 315 articles as *the total dataset* in the rest of this document.

We split the dataset into three subsets in the article level; 80% of the articles are used for training, 10% for validation, and 10% for a test, which corresponds 251, 32, and 32 articles respectively. Overall, on the sentence level, the training set contains 3,582 sentences, validation set has 399 sentences, and the test set has 441 sentences in total.

Figure 4.1 shows the ratios of "event" and "non-event" sentence labels over these different datasets and in the total dataset. As can be seen in the figure, in the total dataset, there are 3,060 non-event sentences and 1,299 event sentences, meaning that 70% of the sentences are labeled as "non-event" and 30% as "event". The same ratio

[1]Annotation manual can be found at https://github.com/emerging-welfare/general$_{i}nfo/tree/master/annotation - manuals$, Accessed on 24 January 2020.

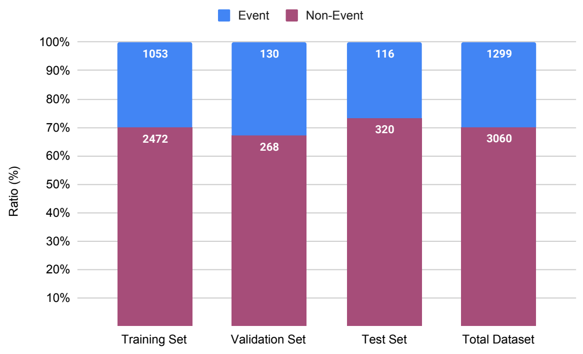


Figure 4.1: The ratios of event and non-event labels over the sentences across different subsets and in the total dataset. The numbers on the chart show the number of sentences belongs to that particular category.

is observed for the training data, with 2,472 non-event and 1,053 event sentences. The validation set and test sets are quite close to this ratio with 67% and 73% “non-event” ratios. While the validation set contains 130 event sentences, the test set contains 116 event sentences (Appendix, Table B.1).

Figure 4.2 shows the ratios of “event” and “non-event” sentences for each article in the total dataset and all its subsets. In 222 articles, more than 50% of the sentences are labeled as “non-event”, while another 75 articles have the “event” sentences as the majority class. Hence, in the 70% of the articles “non-event” sentences are the majority, and 24% of the articles the “event” sentences are the majority. There are also 18 articles with 50%-50% of the labels, which makes the 6% of the articles. Finally, we also see that another six articles contain only “event” sentences. Similarly, Figure 4.3 shows the ratios of “event” and “non-event” sentences for each article in the test set. As we can see in the figure, the distribution of the ratios is similar to the total dataset. Furthermore, in the test set, on average, “non-event” sentences are the majority in the 74% of the articles and “event” sentences are the majority in 21% of them. These statistics are helpful to understand the outcome of using the whole article for the context-dependent classification of the sentences.

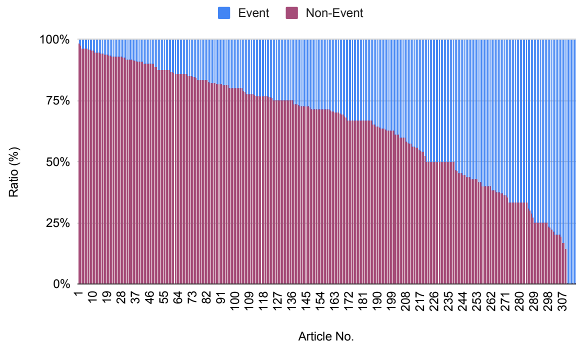


Figure 4.2: The ratios of “event” and “non-event” sentences per article in the total dataset. Each bar is an article. The blue shows the density of event sentences in an article and the red shows the density of non-event sentences in an article.

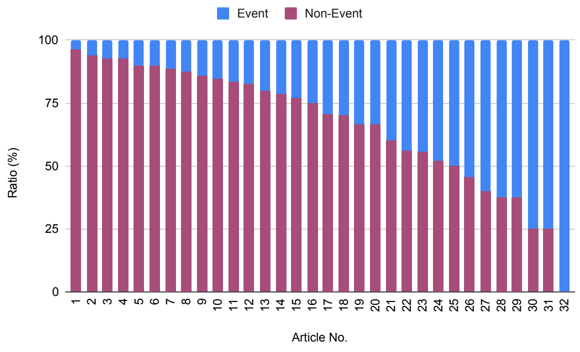


Figure 4.3: The ratios of “event” and “non-event” sentences per article in the test set. Each bar is an article. The blue shows the density of event sentences in an article and the red shows the density of non-event sentences in an article.

Figure 4.4 demonstrates the distributions of different sentence lengths based on token counts in the different datasets. The figures are created to give an idea of the sentence lengths. Although some of our approaches have their own tokenization method (e.g., BERT tokenizer), we tokenize the words by splitting spaces. The maximum number of tokens in a single sentence in the total dataset (also the training set) is 105, while the maximum in the validation and test sets are 97 and 95 (Appendix, Table B.2). There are only 15 sentences longer than 65 token lengths. The average lengths in the total dataset, training set, validation set, and test set are all 24. The median of the sentence lengths in the total dataset, training set, validation set, and test set are 22, 22, 22, 23, respectively. In this project, we are encoding sentences by using embeddings of words. Thus, the number of words have an impact on our results.

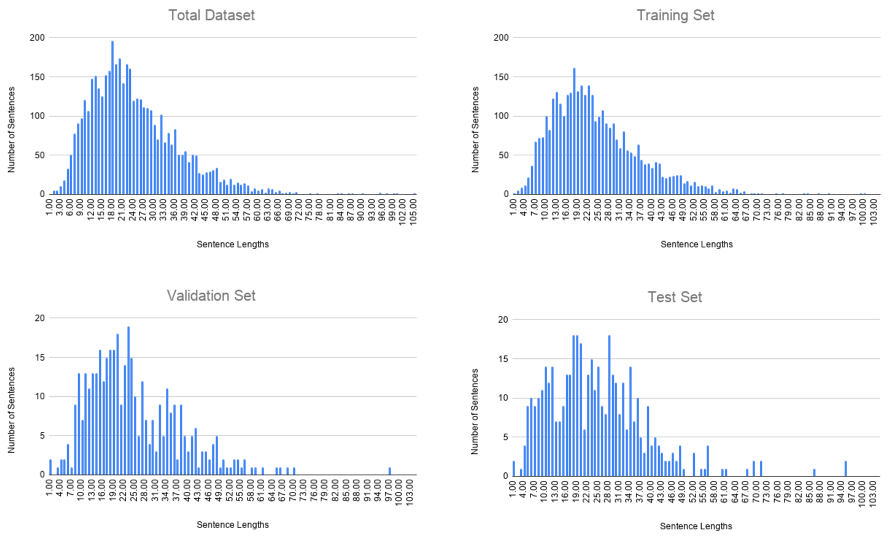


Figure 4.4: The distributions of different sentence lengths based on token counts in different datasets.

4.1.2 Evaluation Metrics

In this study, we use the same evaluation metrics for all experiments. These metrics are precision, recall, and f1-score. The precision metric shows us how precise (or accurate) a model's prediction is on a class. It is the ratio of the number of correctly predicted instances to the number of predictions made:

$$precision = \frac{correct\ predictions}{all\ predictions} = \frac{true\ positives}{true\ positives + false\ positives} \tag{4.1}$$

For instance, in our binary classification case, the precision score is calculated by dividing the correctly predicted "event" sentences by any "event" prediction made by a model.

Recall metric helps us understand how much of relevant data for a class can be correctly predicted by a model. It is calculated by dividing the number of correctly predicted instances by the number of all relevant instances:

$$recall = \frac{correct\ predictions}{all\ correct\ answers} = \frac{true\ positives}{true\ positives + false\ negatives} \tag{4.2}$$

For our binary classification case, a recall score of a model is calculated by dividing the correctly predicted "event" sentences by the number of sentences manually labeled as "event" in the test set. The precision and recall scores are providing us with different views on the performance of a model. As these views are equally crucial for us, we are using a third metric, f1-score, that strikes a balance between them. f1-score is calculated by taking the harmonic mean of precision and recall scores for a class:

$$\begin{aligned} f1 &= 2 \times \frac{true\ positives}{(2 \times true\ positives) + false\ negatives + false\ positives} \\ &= 2 \times \frac{precision \times recall}{precision + recall} \end{aligned} \tag{4.3}$$

As can be seen in Equation 4.3, f1-score ignores the number of true negatives which helps to understand the actual performance of a class. Especially in imbalanced datasets such as ours, where the majority of the instances are true negatives. Each of these metrics is calculated per class both in binary and multi-class setups. However, we aim to generate a single score that can summarize the overall performance of a model covering all the classes. For this, we take an average of the scores calculated per class. There are three different averaging approaches: micro, macro, and weighted averaging. In the micro averaging method, each correct prediction is added in a unified set of true positives, and each error is added to the set of false-positives. Hence, the false-negative cases are treated as same as the false-positives. Consequently, given the formulas, recall (Eq. 4.2), precision (Eq. 4.1), and f1-score (Eq. 4.3) calculation become the same. The weighted averaging approach is the arithmetic mean of the scores per class where each class is multiplied with the number of instances for their corresponding classes before getting divided by the total number of instances. Given that n is the number of classes and w_i is the number of instances for the corresponding class, we can formulate the weighted average as:

$$Weightedf_1 = \frac{\sum_{i=1}^{n} w_i f_{1_i}}{\sum_{i=1}^{n} w_i} \tag{4.4}$$

Finally, the macro averaging is the simple arithmetic mean of the metric for each class. We can formulate macro averaging as following where n is the number of classes:

$$Macrof_1 = \frac{1}{n} \times \sum_{i=1}^{n} f_{1_i} \tag{4.5}$$

The micro and weighted averaging strategies can be misleading about a model's performance if there is a significant class imbalance in the dataset. When the volume of the majority class is too high, a model tends to overpredict the majority class. Thus, the majority class obtains high scores. This situation may happen even if the other classes are not predicted at all. Micro and weighted averaging follow the majority class, and thus, hide the model's actual performance. In this study, we use macro averaged scores to prevent the majority class dominating the others in the evaluation scores caused by our imbalanced dataset. Table 4.1 shows the scores calculated when all of the instances in the test set are predicted as "non-event". In such a case, macro averaged precision, recall, and f1-score are 0.37, 0.50, and 0.42, respectively. Similar behaviour is observed when all of the instances are predicted as "event" which results in 0.13 precision, 0.50 recall, and 0.21 f1-score. We consider these numbers as our lower-bound scores.

	All predicted as "non-event"			All predicted as "event"		
	Precision	Recall	F1-score	Precision	Recall	F1-score
"non-event"	0.74	1.00	0.85	0.00	0.00	0.00
"event"	0.00	0.00	0.00	0.26	1.00	0.42
Macro Averages	0.37	0.50	0.42	0.13	0.50	0.21

Table 4.1: Evaluation metric scores when a model predicts all of the data points as "non-event" and "event" separately.

4.2 Results

In this section, we present the results of the experiments held during the study. We divided the content into two categories, corresponding to the methods introduced in Chapter 3. The first section shows the results of the neural network-based context-agnostic approach compared to the classical machine learning approaches. The second section contains the results from our context-dependent approach compared to the context-agnostic approach results. Additionally, for each approach, we report on the results of our method using the different architecture of our networks and various optimization techniques. Thus, within each section, we first present the results of the optimization experiments, pick the best fitting models and use them in the comparisons.

4.2.1 Context-agnostic Classification Experiments

As explained in Section 3.1.3, we created six different neural network designs. Each architecture is trained with ELMo and BERT models separately. Additionally, each of the models trained once with Adam optimizer and once with RMSprop optimizer. We first present the results from ELMo-based networks and then present the BERT-based network results. Finally, we compare the highest scoring results obtained by neural network models to the results from classical machine learning models.

Architecture Optimization Results

Figure 4.5 shows the macro averaged f1-scores obtained on the validation set by using ELMo pre-trained word embeddings in our neural networks. The figure shows that all models(except for a few) obtain f1-score between 83% and 86%. The highest score was 86.6% obtained by the Model B2 trained with Adam optimizer and 0.005 initial learning rate (green bar on Figure 4.5). We use this model for our comparisons to the other approaches, such as the BERT-based models.

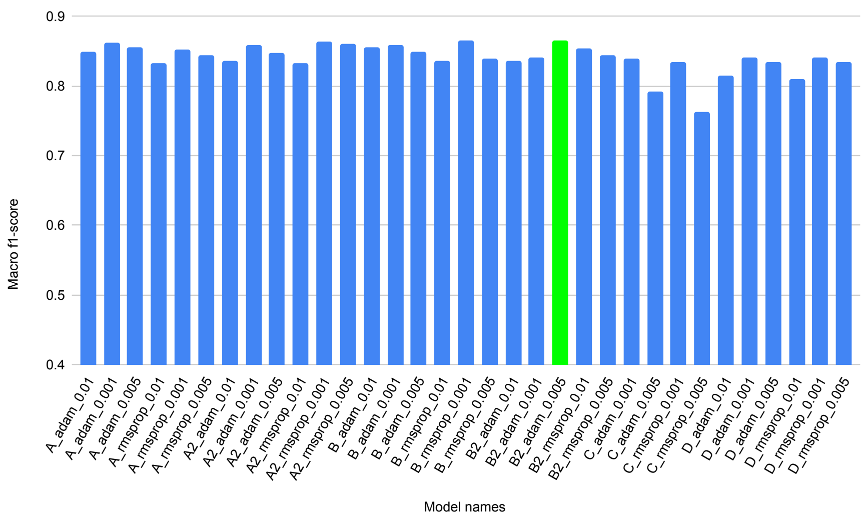


Figure 4.5: Optimization results of different neural network architectures with ELMo embeddings obtained on the validation dataset. The graph only shows the scores higher than 70%. The horizontal axis displays names with the representative value for the architecture, the optimizer, and the learning rate used to train the model. The green bar shows the best performing model.

The macro averaged f1-scores obtained on the validation set by BERT-based neural network models is shown in Figure 4.6. We see that most of the results vary from 83% to 88%. The highest score was 87.9% obtained by the Model A optimized by Adam with 0.0001 learning rate (green bar on Figure 4.6). Thus, we used this model in our comparisons to other approaches.

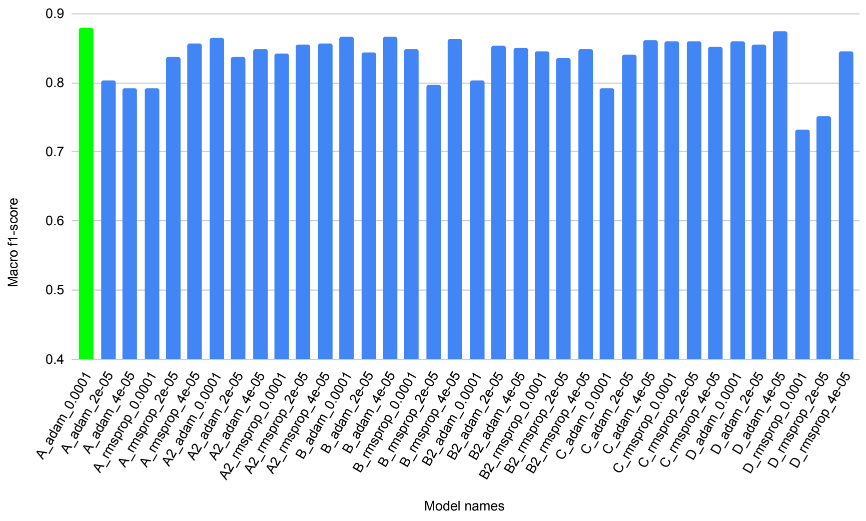


Figure 4.6: Optimization results on the validation set of different neural network architectures with BERT encodings. The graph only shows the scores higher than 70%. The horizontal axis displays names with the representative value for the architecture, the optimizer, and the learning rate used to train the model. The green bar shows the result from the highest scoring model.

Comparing different context-agnostic methodologies

Finally, we compare the highest-scoring artificial neural network-based models to optimized classical machine learning model results. In Table 4.2, we see that neural networks-based approaches outperform classical machine learning models on the validation set. While decision tree, random forest and SVM models obtain at most 82% f1-score, both BERT-based and ELMo-based neural network models' f1-scores are above 85% on the validation set. Likewise, we see that BERT-based Model A network outperform the TF-IDF-based Model A neural network with a high margin on the validation set and the test set.

Amongst the classical machine learning models, the decision tree model obtains the highest f1-score on the test set (79%). The random forest model follows them with a slightly lower score. While the differences between these scores are not marginal, BERT-based Model A obtains the highest f1-score at 82.2% on the test set and outperforms the decision tree model by 4% with respect to f1-score. We observe that ELMo-based Model B2 obtains 78.5% f1-score and does not outperform the decision tree model on the test set.

Here we also see a comparison between feature-based ELMo-based and fine-tuning based (BERT) approaches. BERT-based Model A obtains 87.9% f1-score, ELMO-based Model B2 obtains 86.6% on the validation set. On the test set, BERT-based Model A obtains 82.2%, and ELMO-based architecture Model B2 obtains 78.5% f1-score. Thus, we see that BERT-based models perform better than the ELMO-based models both on the validation and test sets.

	Macro F1-scores (%)	
	On Validation Set	On Test set
Decision Tree (TF-IDF)	81	79
Random Forest (TF-IDF)	82	78
Support Vector Machines (TF-IDF)	80	70
TF-IDF-based Model A	79	72
ELMo-based Model B2	86.6	78.5
BERT-based Model A	87.9	82.2

Table 4.2: Comparisons of context-agnostic approaches with highest scores obtained during optimization.

4.2.2 Context-dependent Classification Experiments

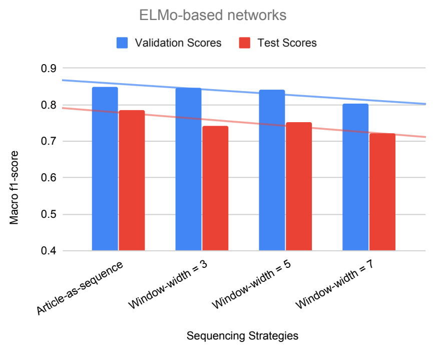
To determine the optimal network architecture for context-dependent neural networks, we defined four different neural network designs which have been trained with ELMo and BERT models separately. Additionally, each of the models trained once with Adam optimizer and once with RMSprop optimizer. These experiments have been repeated for each sequencing strategy and window-width hyperparameter. Here we only display the highest scores obtained for each strategy to make comparisons between them. Finally, we make a comparison with the highest scores for the context-dependent experiments to the results of the context-agnostic experiments.

Sequencing Strategy Results

In Section 3.2.2, we introduced two different strategies to create sequences of sentences: sliding window and article-as-sequence. Each of the network designs has been trained with each of the strategies. Within the sliding window sequencing experiments, we have trained each network design with window-width 3, 5, and 7.

Figure 4.7 shows results obtained by the models scored the highest f1-score on the validation set. Article-as-sequence strategy with ELMo-based networks obtains 78.6% f1-score on the test set, which is the highest score of all context-dependent experiment results. We observe that the ELMo-based models obtain 74.3% f1-scores on the test set with the sliding window strategy (window-width = 3). The BERT-based networks obtain 74.9% f1-score on the test set with article-as-sequence strategy and obtain 66% with the sliding window strategy (window-width = 3). Thus, we observe that the f1-scores are decreased when the sliding-window strategy is applied.

We observe that the highest scores are obtained with article-as-sequence strategy. When we apply the sliding-window strategy, we see a drop in the performance of the networks both on the validation set and the test set. This is easy to spot with the blue (for validation) and red (for test) trend lines given in Figure 4.7. Thus, we have used the highest-scoring model with the article-as-sequence strategy in our further comparisons.



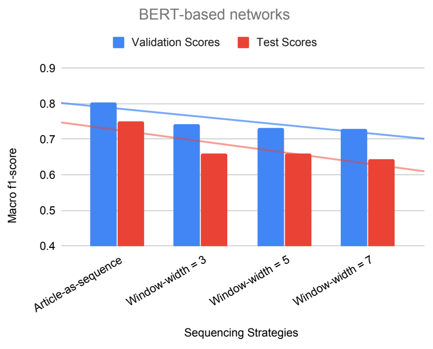


Figure 4.7: Comparisons of the highest scoring models from different sequencing strategies. The left graph shows the results obtained with ELMo-based networks, and the right one shows the results obtained from BERT-based networks. The blue lines show the trend of the scores obtained on validation set and the red lines show the trend of the scores obtained on the test set. Window-width here is the hyperparameter set for sliding window strategy.

Comparing context-agnostic and context-dependent methodologies

Finally, we compare the results of the context-dependent approach experiments to the results of the context-agnostic approach experiments. For that, we have chosen the highest-scoring BERT-based and ELMo-based model results from each of the approaches. Table 4.3 shows these results. We see that context-agnostic BERT-based model outperforms the other models on the test set by obtaining 82.2% f1-score. The context-dependent BERT-model obtains a score as low as 74.9% on the test set. The context-agnostic ELMo-based model obtains 78.5% f1-score, and the context-dependent ELMo-based model obtains 78.6% f1-score, both on the test set.

We see that the ELMo-based models perform close to each other. However, when the two BERT-based models are compared, we see that the context-agnostic model outperforms the context-dependent model with a margin as high as 7.3%.

	Macro F1-scores	
	On Validation Set	On Test set
Context-agnostic ELMo-based (Model B2)	86.6	78.5
Context-agnostic BERT-based (Model A)	**87.9**	**82.2**
Context-dependent ELMo-based (Model G)	84.9	78.6
Context-dependent BERT-based (Model F)	80.2	74.9

Table 4.3: Comparisons between highest-scoring context-dependent approach results and the highest-scoring context-agnostic approach results. Here, both of the context-dependent model results are obtained with the article-as-sequence strategy.

Chapter 5

Conclusions

In this study, we first explored the effect of the recent state-of-the-art pre-trained contextual embeddings on sentence classification in a context-agnostic setup. Second, we introduced a sequential context-dependent approach to the same task and explored different sequencing strategies. We evaluated our models on a real-world dataset which aims to detect sentences related to protest events in newspapers. Here, we summarize our findings and answer the research questions raised in Chapter 1.

Our first research question was *"How can we incorporate pre-trained embeddings to perform sentence classification?"*. In order to answer this research question, we have implemented various neural network-based and classical machine learning-based approaches. We integrated two contextual pre-trained word embedding models (ELMo and BERT) into the neural network models. ELMo pre-trained model was integrated with a feature-based strategy, and BERT pre-trained model was integrated by following a fine-tuning strategy. In conclusions, we showed that the BERT-based neural network approach outperforms all feature-based approaches (using both neural networks and classical machine learning algorithms). Comparing the two contextual word embeddings, we observed that the fine-tuned BERT-based models obtained better results compared to ELMo-based models.

Our second research question was *"Does taking the context of a sentence into account improves sentence classification?"*. To answer this research question, we employed a sequential approach with recurrent neural network architectures that can classify sequences of sentences. We found that the ELMo-based model in context-dependent setting performs as good as ELMo-based context-agnostic models. On the contrary, the BERT-based context-agnostic models outperform context-dependent ones. In conclusions, we did not observe improvements using the context-dependent approach compared to the context-agnostic one. Additionally, in the scope of the context-dependent approaches, we showed that encoding the entire articles as sequences obtains better result compared to encoding each sentence with its neighbouring sentences in a sliding-window approach.

Future Work

During the study, we have used the average pooling method to encode sentences from word embeddings. Average pooling takes each word with equal weight. However, each word in a text has its own importance. In a future study, we want to experiment with building an attention mechanism that assigns different weights to words based on their importance and encode sentences by taking these weights into account.

We have used two different strategies to implement the context-dependent approach. One of them was a many-to-many sequence classification that takes an entire article as a sequence, and the other was a many-to-one sequence classification where each sequence is created by a sliding window moving over the sentences. As a future study, we plan to experiment with a hybrid approach, where a many-to-many classification is applied to the sequences created by a sliding window. This would generate more than one label for each sentence, where we can ensemble these labels to obtain the final label.

We also plan to design more complex context-dependent networks, especially for the sentence encoding part. For instance, after the average pooling layer, we can add convolutional layers prior to the sequence classification with LSTM layers. Furthermore, as a future experiment, we plan to experiment with using the BERT word embeddings as feature vectors of classical machine learning algorithms.

Finally, we could fine-tune only four layers of BERT language models due to the limitations of the hardware. In a future study, we plan to experiment with fine-tuning every layer of the BERT model.

Acknowledgements

This work is funded by European Research Council through the ‘The New Politics of Welfare: Towards an Emerging Markets Welfare State Regime’ project with project number 714868 and proposal acronym ‘EmergingWelfare’.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth Trippe, Juan Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. 07 2017.

[3] Nicolas Audebert, Catherine Herold, Kuider Slimani, and Cédric Vidal. Multimodal deep networks for text and image-based document classification. In *Conférence Nationale sur les Applications Pratiques de l'Intelligence Artificielle (APIA)*, Toulouse, France, July 2019.

[4] Iñigo Barandiaran. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell*, 20(8):1–22, 1998.

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[6] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, Jan 1992.

[7] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 12 2013.

[8] François Chollet et al. Keras. `https://keras.io`, 2015.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Jesse Hammond and Nils B Weidmann. Using machine-coded event data for the micro-level study of political violence. *Research & Politics*, 1(2):2053168014539924, 2014.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[15] Martin Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26, 2006.

[16] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. 08 2015.

[17] Ali Hürriyetoğlu, Erdem Yörük, Deniz Yüret, Çağrı Yoltar, Burak Gürel, Fırat Duruşan, Osman Mutlu, and Arda Akdemir. Overview of clef 2019 lab protestnews: extracting protests from news in a cross-context setting. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 425–432. Springer, 2019.

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

[19] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 364–371. IEEE, 2017.

[20] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[23] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 515–520, San Diego, California, June 2016. Association for Computational Linguistics.

[24] Yichen Li, Arvind Tripathi, and Ananth Srinivasan. Challenges in short text classification: The case of online auction disclosure. In *MCIS*, page 18, 2016.

[25] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.

[26] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[27] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[28] Martina Naughton, Nicola Stokes, and Joe Carthy. Sentence-level event classification in unstructured texts. *Information retrieval*, 13(2):132–156, 2010.

[29] Martina Naughton, Nicola Stokes, and Joe Carthy. Sentence-level event classification in unstructured texts. *Information retrieval*, 13(2):132–156, 2010.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[32] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[33] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, pages 91–100. ACM, 2008.

[34] Kemal Polat, Salih Güneş, and Ahmet Arslan. A cascade learning system for classification of diabetes disease: Generalized discriminant analysis and least square support vector machine. *Expert systems with applications*, 34(1):482–487, 2008.

[35] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[36] PC Rafeeque and S Sendhilkumar. A survey on short text analysis in web. In *2011 Third International Conference on Advanced Computing*, pages 365–371. IEEE, 2011.

[37] Lior Rokach and Oded Z Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[39] S. S. and J. M. *Neural Networks for Natural Language Processing.* Advances in Computer and Electrical Engineering. IGI Global, 2019.

[40] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, March 2008.

[41] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[42] Ge Song, Yunming Ye, Xiaolin Du, Xiaohui Huang, and Shifu Bie. Short text classification: A survey. *Journal of multimedia*, 9(5):635, 2014.

[43] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[44] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842. ACM, 2010.

[45] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[46] Bing-kun Wang, Yong-feng Huang, Wan-xia Yang, and Xing Li. Short text classification based on strong feature thesaurus. *Journal of Zhejiang University SCIENCE C*, 13(9):649–659, 2012.

[47] Jin Wang, Zhongyuan Wang, Dawei Zhang, and Jun Yan. Combining knowledge with deep convolutional neural networks for short text classification. In *IJCAI*, pages 2915–2921, 2017.

[48] Wei Wang, Ryan Kennedy, David Lazer, and Naren Ramakrishnan. Growing pains for global monitoring of societal events. *Science*, 353(6307):1502–1503, 2016.

[49] Tingmin Wu, Shigang Liu, Jun Zhang, and Yang Xiang. Twitter spam detection based on deep learning. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17, pages 3:1–3:8, New York, NY, USA, 2017. ACM.

[50] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 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, June 2016. Association for Computational Linguistics.

[51] SHI Yong-feng and Zhao Yan-ping. Comparison of text categorization algorithms. *Wuhan university Journal of natural sciences*, 9(5):798–804, 2004.

[52] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[53] Zheng Zhu. Improving search engines via classification. *University of London*, 2011.

Appendix A

Neural Network Architectures

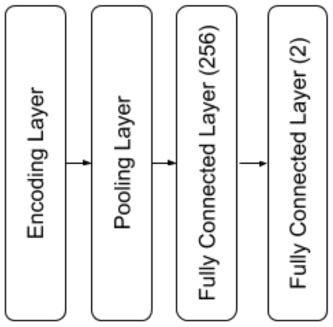


Figure A.1: Context-agnostic Model A Architecture.

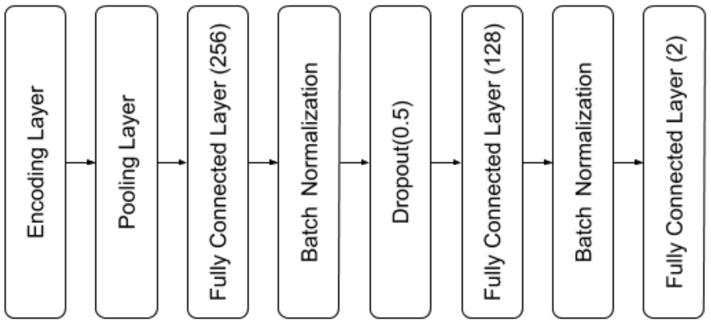

Encoding Layer
Pooling Layer
Fully Connected Layer (256)
Batch Normalization
Dropout(0.5)
Fully Connected Layer (128)
Batch Normalization
Fully Connected Layer (2)

Figure A.2: Context-agnostic Model A2 Architecture.

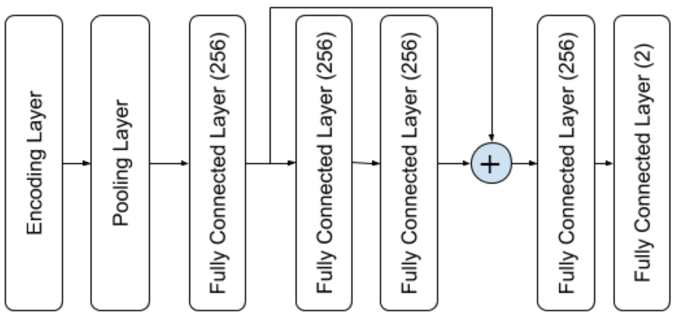

Encoding Layer
Pooling Layer
Fully Connected Layer (256)
Fully Connected Layer (256)
Fully Connected Layer (256)
+
Fully Connected Layer (256)
Fully Connected Layer (2)

Figure A.3: Context-agnostic Model B Architecture.

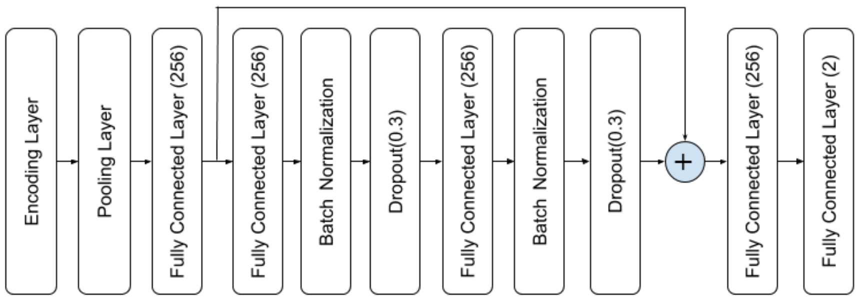

Encoding Layer
Pooling Layer
Fully Connected Layer (256)
Fully Connected Layer (256)
Batch Normalization
Dropout(0.3)
Fully Connected Layer (256)
Batch Normalization
Dropout(0.3)
+
Fully Connected Layer (256)
Fully Connected Layer (2)

Figure A.4: Context-agnostic Model B2 Architecture.

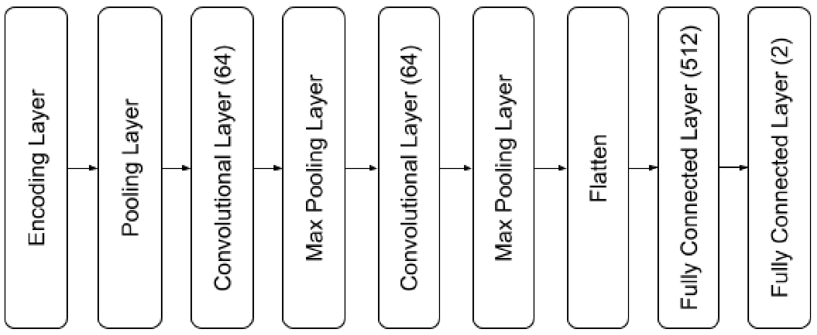


Figure A.5: Context-agnostic Model C Architecture.

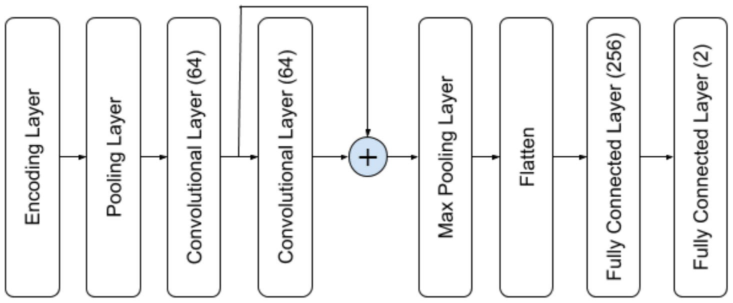


Figure A.6: Context-agnostic Model D Architecture.

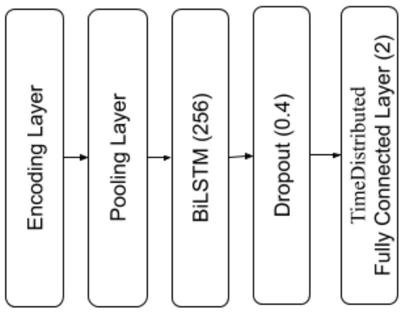


Figure A.7: Context-dependent Model E Architecture.

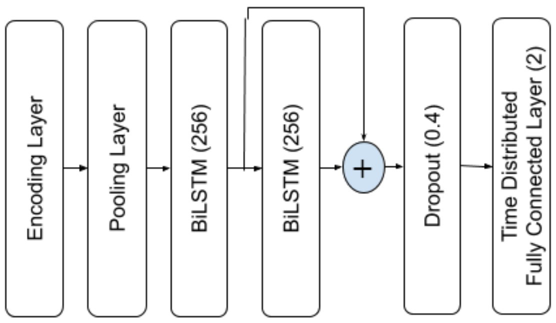


Figure A.8: Context-dependent Model F Architecture.

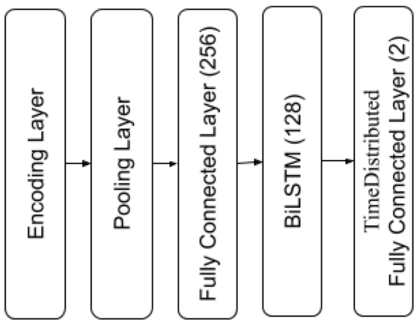


Figure A.9: Context-dependent Model G Architecture.

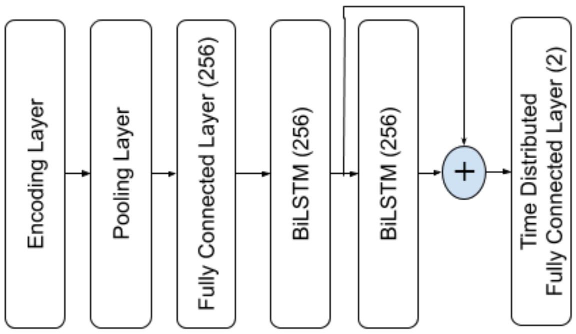


Figure A.10: Context-dependent Model H Architecture.

Appendix B

Dataset

	Non-Event	Event
Training Set	2,472 (70%)	1,053 (30%)
Validation Set	268 (67%)	130 (33%)
Test Set	320 (73%)	116 (27%)
Total Dataset	3,060 (70%)	1,299 (30%)

Table B.1: The ratios of event and non-event labels over the sentences across different subsets and in the total dataset.

	Maximum Sentence Lengths	Average Sentence Lengths
Training Set	105	24
Validation Set	97	24
Test Set	95	24
Total Dataset	105	24

Table B.2: Maximum and average sentence lengths based on tokens across different subsets and in the total dataset.