Semantic Subject Indexing: Comparison of Deep Learning Methods on Titles and Full-Texts

Masterarbeit

Christian-Albrechts-Universität zu Kiel
Technische Fakultät
Institut für Informatik
AG Knowledge Discovery

angefertigt von:	**Florian Mai**
Matrikelnummer:	006790
betreuender Hochschullehrer:	Prof. Dr. Ansgar Scherp
Zweitkorrektor:	M.Sc. Lukas Galke

Kiel, den 9. Februar 2018

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig angefertigt, nur die angegebenen Hilfsmittel benutzt und wörtlich oder dem Sinne nach den Quellen entnommene Stellen als solche gekennzeichnet zu haben.

Die Arbeit hat noch nicht zum Erwerb eines anderen Scheins bzw. als eine andere Prüfungsleistung vorgelegen.

Kiel, den 9. Februar 2018

Florian Mai

Contents

List of Figures

List of Tables

Semantic Subject Indexing: Comparison of Deep Learning Methods on Titles and Full-Texts

Florian Mai
Kiel University
stu96542@informatik.uni-kiel.de

ABSTRACT

For (semi-)automated subject indexing systems in digital libraries, it is often more practical to use metadata such as the title of a publication instead of the full-text or the abstract. Therefore, it is desirable to have good text mining and text classification algorithms that operate well already on the title of a publication. So far, the classification performance on titles is not competitive with the performance on the full-texts if the same number of training samples is used for training. However, it is much easier to obtain title data in large quantities and to use it for training than full-text data. In this paper, we investigate the question how models obtained from training on increasing amounts of title training data compare to models from training on a constant number of full-texts. We evaluate this question on a large-scale dataset from the medical domain (PubMed) and from economics (EconBiz). In these datasets, the titles and annotations of millions of publications are available, and they outnumber the available full-texts by a factor of 20 and 15, respectively. To exploit these large amounts of data to their full potential, we develop three strong deep learning classifiers and evaluate their performance on the two datasets. The results are promising. On the EconBiz dataset, all three classifiers outperform their full-text counterparts by a large margin. The best title-based classifier outperforms the best full-text method by 9.4%. On the PubMed dataset, the best title-based method almost reaches the performance of the best full-text classifier, with a difference of only 2.9%.

CCS CONCEPTS

• **Information systems → Digital libraries and archives**; • **Computing methodologies → Neural networks**; *Natural language processing*;

KEYWORDS

Text classification; deep learning; digital libraries

1 INTRODUCTION

Semantic annotations are crucial for users of digital libraries as they enhance the search of scientific documents. Given the large amount of new publications [11], automatic annotation systems are a useful tool for human expert annotators working at digital libraries to classify the publications into categories from a (hierarchical) thesaurus. However, providing automated recommendations for subject indexing in such systems is a challenging task. This is partly due to the data from which recommendations may be generated. Often neither the full-text of a publication nor its abstract may be available. For instance, the digital library EconBiz contains only for 15% of the documents an abstract. Even when the content can be legally provided by the library to the end users, copyright laws or regulations of the publishers may prevent text mining. Moreover, collecting and processing PDFs where it is possible, e. g., for some Open Access documents, adds high computational requirements to the library. This puts annotation methods on demand that are based on data with better availability, such as the title. Previous work by Galke et al. [24], however, has shown that title-based methods considerably fall behind full-text methods in terms of performance when the number of samples for training is equal. If our classifier was a human expert, this would not be a surprising result. A full-text contains more information and therefore also more indication of the publication's topic. A human expert will always make better annotations based on the full-text. In fact, the annotations that are used as gold-standard for automated subject indexing experiments are often created based on the full-text.

However, we argue that machine learning algorithms work differently than a human. In contrast to a human, they often require hundreds of thousands or even millions of training data to yield satisfactory models [12]. These amounts of data are not always available in the real world. One common reason is that human expertise is required for creating a large enough gold standard, which is expensive. For semantic subject indexing, the availability issues mentioned above do not only come into play at prediction time, i.e., when a machine learning model is used in a productive system, but also during training. In effect, methods based on the full-texts have drastically less training data available than methods based on titles. This raises the question if title-based methods can potentially narrow the performance gap to full-text methods by fully incorporating all training data available.

In this paper, we address this question. Formally, semantic indexing is framed as a multi-label classification problem, where a (commonly small) subset of labels has to be selected from a (relatively large) set of labels. From two digital libraries of scientific literature, PubMed and EconBiz, we have compiled an English full-text dataset and an English title dataset. Our compiled datasets are quite different with respect to their size. From PubMed, we extracted 12.83 million titles, for 5% of which a full-text is available (646k). From EconBiz, we extracted 1.06 million titles, of which approximately 7% have a full-text (71k). In order to fully utilize these large amounts of data, we develop and compare three different classifiers that have emerged from the deep learning community in recent years. Deep learning has advanced the state-of-the-art in many fields, such as vision, speech, and text [47]. These techniques are known to shine when a lot of training data is available. For text classification in particular, recent work [101] suggests that deep learning starts to outperform strong traditional models when 650k or more training samples are available. The number of full-text in PubMed is right at the edge of this number, whereas EconBiz has far less full-texts,

making these datasets an interesting and revealing choice. In natural language processing, different types of neural networks have been successfully employed on different tasks, but it is an open question whether convolutional neural networks (CNNs), recurrent neural networks (RNNs), or multi-layer-perceptrons (MLPs) are superior for text classification tasks. Therefore, we employ a representative of each type in our study. We compare them against another strong MLP baseline (*Base-MLP*), which has previously been shown to also outperform traditional bag-of-words classifiers such as SVMs, Naive Bayes, and kNN [24]. Since the label space in our datasets is very large, our study can be understood as "extreme multi-label classification". Here, only few studies have leveraged deep learning techniques to tackle the considerably harder problem when the label space is large [51, 100]. Hence, with our study, we contribute to the knowledge in this field, as well.

The results of our study indicate that title-based methods can match or even outperform the full-text performance when enough training data is available. On EconBiz, the best title classifier (MLP) performs on par with the best full-text classifier (MLP) when only 8× as many titles are used for training than for full-text. When all available titles are used (approximately 15× more than full-texts), the title-based MLP outperforms its full-text counterpart by 9.4%. On the PubMed dataset, the best title method is the RNN, and it almost reaches the best full-text performance produced by an MLP. The gap is only as small as 2.9%. When using the same number of titles as full-texts are available, the gap in classification performance is 10.7%, indicating a considerable benefit from leveraging all title data. Generally, the MLP performs well, outperforming RNN and CNN in all but one case. It also consistently outperforms the baseline when all titles or full-texts are used. Moreover, our analysis suggests that our proposed classifiers are well-chosen for our study, because they benefit from increasing amounts of training data better than the baseline. The RNN performs rather poor on full-text, but shows strong performance on titles. While also yielding reasonably good results with CNN, it performs clearly below the other classifiers in all cases. This is a surprise, because a lot of the recent literature on large-scale text classification has focused on CNNs (e.g. [17, 46, 101], also see Section 2). Thus, our results indicate that it may be worth it to shift the research focus more towards other types of neural networks for text classification.

Our contributions can be summarized as follows:

- For the first time, we study the practically relevant question whether title-based methods can reach the performance of full-text-based methods by exploiting the surplus of available training data.
- We demonstrate that title-based methods are on par or even outperform full-text methods when the number of training samples is sufficiently large.
- We develop and compare three strong classifiers for (extreme) multi-label text classification, contributing to the debate on which type of neural network is superior for text classification.

The remainder of the paper is structured as follows. In the subsequent section, we review relevant work on the comparison of short texts and full-text and on (deep) text classification. In Section 3, we describe our deep learning models. We introduce the datasets and experimental setup in Section 4, and present the results in Section 5. We analyze and discuss the results in Section 6, before we conclude.

2 RELATED WORK

In this section, we review previous literature relevant to our study. First, we discuss papers that compare performance on titles with performance on full-text. Next, we briefly discuss semantic subject indexing and methods for multi-label text classification other than deep learning. Finally, we discuss current deep learning methods for text classification.

Title versus Full-Text. The work directly related to our study is the one by Galke et al. [24]. The authors compare titles with full-texts for multi-label text classification on four datasets. Two datasets consist of scientific publications and are therefore comparable to the datasets in this study. In their experiments, the authors used the same number of samples for the title-based methods and the full-text methods. They found that the title-based methods can yield reasonably good performance. However, the difference between title and full-text on the two scientific datasets is still 10% and 20% in favor of the full-text, respectively. The first dataset is from the economics domain, and it is an earlier version of the one used in this study. The second dataset is from the political sciences. In their comparison of classifiers, an MLP to which in this study we refer as *Base-MLP*, outperforms all other (non-neural) classifiers in 7 out of 8 cases. All the presented classifiers are based on the bag-of-words (BoW) feature representation, a traditionally strong baseline for text classification that disregards word order. Due to clearly superior performance, Base-MLP can also be considered the best representative of traditional BoW models. We therefore report its performance as our baseline for all subsequently developed models.

The comparison of metadata vs. full-texts has also been studied for tasks other than document classification. Nascimento et al. [63] studied how well queries for web search can be generated from title, abstract, or body, respectively. These queries are then issued to Web information sources to retrieve papers for recommendation. In their experiments, abstracts yielded the best queries, closely followed by the body. The title clearly performs worse. On the contrary, Nishioka and Scherp [64] have demonstrated that paper recommendations to researchers based on their Twitter profile can be made using only the title of the paper. This is achieved by their novel profiling method HCF-IDF, which is able to extract sufficient conceptual information from the title through spreading activation over a hierarchical knowledge base. Galke et al. [25] use text embedding techniques for the information retrieval task and compare how well these techniques perform when the index is built upon the title, abstract, or full-text of the documents. Here, titles have demonstrated a clear advantage over abstract and full-text. Lastly, Hemminger et al. [33] compare full-text search with metadata search in the PubMed database, where full-text search yields better results.

Semantic Subject Indexing. The task of subject indexing is to classify a document with terms (also *labels*) that describe the subject of the document. These assigned terms are often used to enable search in a retrieval system. Depending on the type of indexing, the terms may be extracted from the document itself or may be

taken from a controlled vocabulary [16]. They may be chosen by the authors of the document, by a trained expert, or even by an automatic system. The research on automatic subject indexing is plentiful, and an overview is provided in the supplementary materials in Section A.1. In this paper, we study the case where the indexing terms come from a thesaurus. In this case, the subject indexing task can be formally framed as multi-label classification, which is discussed next.

Multi-label Text Classification. Text classification is a well-studied problem. k-Nearest-Neighbors and SVM are common choices for text classification (see e.g. [30, 40, 73]). However, kNN's complexity grows in the number of training samples, which is problematic for the training sample sizes we consider in our study. SVMs do not provide a natural adaptation for multi-label classification. A binary relevance classification scheme, however, is impractical when the number of labels is large.

An active field of multi-label text classification research is the MeSH indexing community. This area is concerned with annotating PubMed articles with medical subject headings. BioASQ [88] is a challenge that recently finished its 5th iteration. It has the goal to advance the state-of-the-art in MeSH indexing, and to this end provides large datasets for training. We acknowledge the significance of the MeSH indexing community, and in our study, we include the dataset from the latest iteration of the BioASQ challenge. However, many of the approaches successful at this challenge, including learning to rank [38, 52, 54] and pattern matching [58], make use of features tailored to the biomedical domain. Since in this paper, we study domain-independent methods for subject indexing in digital libraries, these methods are not appropriate.

Deep Learning for Text Classification. Some early works leverage neural networks for multi-label text classification. In order to capture label inter-dependencies, Zhang and Zhou [98] employ a pairwise ranking loss function for text classification. Nam et al. [61] show that replacing the ranking-loss with cross-entropy leads to faster convergence and overall better prediction performance. Furthermore, they are the first to incorporate some of the milestone advancements from the deep learning era like rectified linear units, dropout, and the smart optimizer AdaGrad.

In recent years, neural networks have become the state-of-the-art in multi-class text classification, outperforming traditional linear BoW models, in particular on very large datasets. On a diverse set of text classification datasets of rather small scale (up to 10.7k samples), neural networks have shown their capability to perform well. These include CNNs [42, 95] and RNNs [83], as well as a combination of both [99].

Zhang et al. [101] were the first to introduce several large-scale multi-class text classification datasets ranging from 120k to 3.6 million training samples. The number of classes ranges from two to 14. Zhang et al. proposed a deep, character-based convolutional neural network and compared it with a number of traditional models, including multinomial logistic regression based on bag-of-ngrams with TF-IDF, and deep learning models such as a Long Short-Term Memory network (LSTM) and word-based CNNs. The finding most relevant to our study is that traditional models tend to outperform the neural network architectures on the four relatively small datasets with 560k training samples or less, whereas on the remaining four datasets with 650k training samples or more their neural network approach is superior. This result is the main reason why we choose to employ deep learning techniques in our study.

Several studies employing deep learning on these datasets have followed. Conneau et al. [17] draw inspiration from the computer vision community and improve the performance of character-based CNNs by increasing the depth. Le et al. [46] put these results into perspective by demonstrating that a shallow, word-based CNN performs on par with these models or better. Therefore, we have limited our study to shallow, word-based CNNs. For text classification, it is an open question whether CNNs or RNNs are superior [94]. Consequently, LSTMs had also some success on these large-scale datasets [93, 96], as well as hybrid approaches [92]. Joulin et al. [28] demonstrated that even a linear MLP classifier can yield results competitive with non-linear deep learning methods while maintaining computational efficiency.

To the best of our knowledge, each of the neural network types MLP, CNN, and LSTM provide the current state-of-the-art on at least one of these large-scale datasets (as shown in the work by Le et al. [46]). Hence, with our study we would like to contribute to this discussion by employing a representative of each of the neural network types and employ it for classification on text.

Finally, due to the number of labels in our datasets, our study may be classified as extreme multi-label classification. To the best of our knowledge, only two papers studied the application of deep learning to this task [51, 100], from which we draw some inspiration for our models, which are described below.

3 METHODS

In this section, we present three neural network architectures for text classification used in our study. In the design phase, we aimed at carefully developing the strongest representative of each of the most common types of neural networks, MLPs, CNNs, and RNNs. For transparency, intermediate results of the design phase can be found in the supplementary materials in Section C. Here, we only present the final architectures which are used in the experiments presented in Section 4. These architectures may differ depending on the dataset and type of text (title or full-text) they operate on. This is necessary to avoid a bias for either of the datasets or types of text. The output layer as well as the training procedure in our neural networks are the same for all neural network architectures presented here.

Below, we describe the training procedure, before we go into each neural network architecture up to the last hidden layer. For brevity, we omit a formal mathematical description of the models here. Instead, these are provided in the supplementary materials (Section B.2).

3.1 Training Procedure

The semantic annotation task is formally a *multi-labeling problem*, where instead of belonging to exactly one class, each publication is assigned a set of labels. This is an important difference to most of the previous literature on text classification with large datasets. *Binary Relevance* is a common technique to adapt a classifier for a multi-labeling problem. However, this is a costly technique when

the number of labels is high because it requires to train as many classifiers as there are labels. Neural networks have a more natural way to deal with multi-label classification, which is also made use of by Galke et al. [24]. For multi-class text classification, the softmax activation function is used at the output layer to obtain a probability distribution over the classes. For multi-label classification, however, the sigmoid activation can be employed to determine a probability p_l for each label l whether it should be assigned or not. The difference is that softmax regards all labels at once, while sigmoid makes an independent decision for each label. Finally, the binary decision whether label l is assigned is made by checking whether p_l exceeds a threshold θ.

Using Adam [43], the networks are trained as to minimize the sum of all binary cross-entropy losses over all labels. This has shown to be superior to a ranking-based loss on multi-label text classification [51, 61]. Training is executed in mini-batches of size 256. We employ early stopping for regularization and as criterion to terminate training. Early stopping is performed as follows. After a fixed number $\tilde{b}$ (the *validation frequency*) of mini-batch updates, the model's performance on a validation set in terms of sample-based F_1 measure is evaluated. After each evaluation, the model's weights are saved if the validation score has improved. Training is terminated when the validation score has not improved over the best reported score for 10 consecutive evaluations. For prediction, we load the weights from the best evaluation step. Furthermore, $\tilde{b}$ is set to the minimum of one epoch and 2,000 mini-batch updates. This is necessary because on the largest datasets it can happen that the model both has a point of considerable underfitting and overfitting within the same epoch. Hence, the point where the model generalizes well may be missed. Thus, a sufficiently small $\tilde{b}$ was determined experimentally (see Section C.2.2).

Since the output of the sigmoid activation function can be interpreted as the probability whether a label should be assigned, a typical choice for the threshold is $\theta = 0.5$. However, depending on the evaluation metric, dataset, or model that generates the assignment probabilities, this value does not necessarily yield optimal results. Unfortunately, finding a good value for θ can be computationally expensive, especially when the datasets are very large. Therefore, we use a heuristic that continuously adjusts the threshold *during* training. To this end, the evaluation on the validation set used for early stopping is also used to optimize θ.

Formally, we initially set $\theta_0 := 0.2$, which Galke et al. [24] found to be a better threshold value than 0.5. After each validation step i, where the classifier predicts a probability for each of the $|L|$ labels and each of the n samples in the validation set, accumulated in $P_i \in (0,1)^{n \times |L|}$, we set

$$\theta_i := \underset{\tilde{\theta} \in \{-k*\alpha+\theta_{i-1}, \ldots, k*\alpha+\theta_{i-1}\}}{\arg\max} F_1(P_i; \tilde{\theta})$$

where $\alpha > 0$ is the step size and k controls the number of threshold values to check. This heuristic is based on the observation that the optimal choice for θ_i is in most cases in close proximity to the optimal choice of the previous evaluation step, θ_{i-1}. More detail is provided in Section B.4.3. Since computing the F_1 score can be costly, we set $k = 3$ and $\alpha = 0.01$ to trade off granularity with speed.

In our preliminary experiments (see Section C.2.2), this way of optimizing the threshold consistently yields good results, sometimes even better than when it is optimized manually. We have also experimented with another multi-labeling technique that does not use a threshold. In particular, we have developed a neural version of MetaLabeler [84]. A formal description of the algorithm is provided in the supplementary materials in Section B.3.2. However, since this technique again introduces a hyperparameter that needs dataset-specific tuning (for details, please see Section C.7), we decided against using it and leave a thorough examination of domain-specific hyperparameters for future work.

3.2 Multi-Layer-Perceptron

Our baseline is a multi-layer-perceptron (MLP) described by Galke et al. [24]. It has one hidden layer with 1,000 units and rectifier activation and it takes a TF-IDF [78] bag-of-unigrams as input. The bag-of-unigrams only contains the 25,000 most common unigrams, which we determined to be sufficient for the multi-labeling task (see Section C.2.1 for details). For regularization, dropout [82] is applied after the hidden layer with a keep probability of 0.5. We will refer to this baseline as *Base-MLP*.

We extend the MLP from Galke et al. [24] by incorporating some techniques inspired from recent deep learning literature. The MLP architecture introduced in this study can be viewed as a non-linear adaptation of fastText [28] for multi-label classification. FastText is a popular text classification library that excels at training speed while still attaining classification performance close to state-of-the-art. Fast training is obtained through two major design decisions. First, the costly softmax operation was approximated through hierarchical softmax. Since in this study we deal with a multi-labeling problem and thus employ sigmoid, this is not a concern for our models. Secondly, they employed an efficient linear BoW model that they enhance with feature sharing and local word order information. The feature sharing component is introduced through a hidden layer with identity activation function (in order to retain a linear classifier). The outputs at the hidden layer are then latent features shared among all classes. This architecture of fastText is already similar to our model, except that we employ a non-linear activation function (rectifier). However, for our study, computational speed is not of essence. Moreover, we found experimentally (Section C.5.2) that omitting this non-linearity in fact hurts the classification performance considerably. In fastText, local word-order information is provided in the form of bi-grams. For our models, we integrate this information by adding the 25,000 most common bi-grams in addition to the 25,000 most common unigrams.

In the introduction, it was mentioned that deep neural networks excel when the number of training samples is very large. This is because the representational power of neural networks increases as the number of parameters increases. This can be obtained by adding more layers or by adding more units to the existing layers. Additionally, deeper networks may be able to learn hierarchical representations of the input, as can be observed in the vision domain [32]. The problem with deep neural networks is that they are generally harder to train due to the vanishing and exploding gradient problems. In order to alleviate this problem, a lot of techniques have been proposed that attempt to normalize the output of a layer,

so the respective gradient is well-behaved. Among those techniques, we picked the well-established Batch Normalization [39] and the recently proposed Self-Normalizing Neural Networks [44] and have applied them to our deep MLPs.

In our experimental evaluation, we found wider networks to do better than deeper networks in most cases. The only exception is the PubMed title dataset, where a two-layer neural network with Batch Normalization performs best. For a thorough analysis, please refer to Section C.6.1.

3.3 Convolutional Neural Network

We present a CNN architecture whose core was introduced by Kim [42] for sentence classification and has since been repeatedly adopted and enhanced upon. We adopt and combine some of these enhancements for our model. Details on the effect of our design decisions may be found in Sections C.5.3 and C.6.2.

The CNN operates on word embeddings, which are initialized with a pretrained model but finetuned during training. As in Kim's model, our CNN applies a 1D-convolution by sliding a window over the text in order to extract features at each position. These outputs are then transformed by a non-linear activation function (the detector). Commonly, the most salient position is selected by applying max-pooling after the detector stage. However, Liu et al. [51] instead split the output of the convolution into p nearly equal chunks, and perform max-pooling on each chunk. Afterwards, the outputs of the pooling stages are concatenated. For $p = 1$, this is identical to Kim's architecture. In our experiments, using $p > 1$ improved performance only on full-text. This is comprehensible, because titles are already quite short, so they do not need to be split in chunks.

Commonly, this process is repeated for multiple window sizes. The outputs of these processes are then concatenated before passing them to the next layer. For example, Kim's CNN uses window sizes 3, 4, and 5, while Liu et al. use 2, 4, and 8. We experimentally determined that using 2, 3, 4, 5, and 8 yields to even better results.

In Kim's model, the concatenated output of the pooling stages is directly propagated to the output layer. Liu et al. argue, however, that it is better to have an additional fully-connected layer, called the *bottleneck layer*, with n_b units, where n_b is smaller than the output of the convolutional layer, injected before the output layer. The reasoning is twofold: First, a more narrow layer after the pooling actually reduces the number of parameters in the model if the number of output labels is large. Secondly, it adds more representational power to the network through the increased depth. For our model, we also found that a bottleneck layer is beneficial.

Considering the complexity of our datasets and the number of training samples available, the question of increased capacity arises with CNNs. Similar to MLPs, an increase in capacity can be achieved either through wider convolutions (larger feature-map) or additional stacked layers of convolutions. Since a recent study by Le et al. [46] has shown that depth does not yield improvement over shallow nets, we only consider the former approach. Previously, a feature map size of around 100 was a common choice [42, 46, 51]. On our datasets, we often found a size of up to 400 to considerably improve the results, even on the full-texts of EconBiz, which has relatively few samples.

3.4 Recurrent Neural Network

The Recurrent Neural Network (RNN) is a family of neural networks that was specifically designed for sequential input data. By maintaining a hidden state, the network is able to keep track of previous inputs. However, the vanilla RNN has difficulties keeping track of inputs that are far in the past. Some solutions to this problem have been proposed, most prominently the LSTM [34] and the Gated Recurrent Unit (GRU) [15]. Both architectures base on the principle that they explicitly model the control over whether the current hidden state shall be forgotten, updated, or kept. Both the LSTM and GRU perform comparably in many tasks. For this study, initially we use an LSTM that has already achieved good results for text classification in a study by Zhang et al. [101]. This LSTM is the "vanilla" version described in [29]. Our final model, however, incorporates two techniques which have proven useful for NLP tasks in recent years, attention and bidirectionality [97]. For details on the effect of these techniques, please refer to Section C.5.4.

Since any RNN produces an output at every time step, the outputs have to be aggregated after processing the entire sequence, in order to pass a vector of fixed size to the next layer. We experimented with choosing the last output, computing the sum, computing the average, and computing a weighted average where the weights are determined by an attention mechanism as used by Yang et al. [93]. While the benefit over the other aggregation methods is not large for titles, the attention mechanism consistently performs best. On full-texts, however, the difference is considerable. This is intuitive, because there is less need to focus on specific parts of the input if the input is as short as in a title.

In a standard RNN, the sequence is read from left to right. That means that at any time step i the RNN's state does not encode information that follow after step i, even though those may be critical for understanding the current step of the sequence. Bidirectional RNNs are a simple yet effective way to alleviate this problem. Here, one standard RNN reads the sequence from left to right, and another RNN reads it from right to left. The outputs of both RNNs at each time step are then simply concatenated. Bidirectional RNNs are a common technique to boost the performance over standard RNNs. For text classification in particular, Yang et al. [93] have employed a bidirectional GRU. In our study, we employ a bidirectional LSTM, since we found it to improve the performance considerably.

Again, we made some effort to investigate an increase in the capacity of the LSTM to account for the large number of training samples in our datasets. In the past, both increasing the memory cell size (the width) and stacking LSTMs on top of each other has been successful in some NLP tasks. For text classification, this has not been the case. The results of our experiments support this, where wider LSTMs are superior to stacked LSTMs, even when variational dropout [23] is used. The details of our analysis can be found in Section C.6.3. The single-layer LSTM used by Zhang et al. [101] uses a cell size of 512. For our final experiments, we found cell sizes up to 1,536 work better. Beyond that, the training time becomes unacceptable.

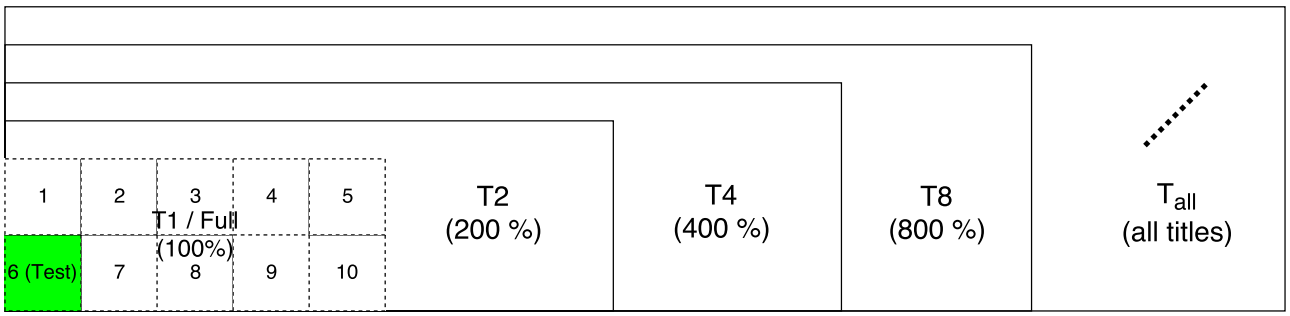


Figure 1: We organize our dataset in several sub-datasets to perform an iterative evaluation. T1 and Full comprise of the same set of publications (all samples where a full-text is available), and is split into 10 folds in order to perform the same 10-fold cross-validation with titles and full-texts for a fair comparison. For T2, T4, T8, increasingly more titles that do not have a full-text are added for training, but are in each cross-validation step evaluated on the same test samples (highlighted in green for exemplification). T_{all} includes all title samples.

4 EXPERIMENTAL SETUP

4.1 Datasets

We built English datasets from two digital libraries of scientific publications. EconBiz[1] is a search portal for economics and business studies. Currently, it contains 2,485,000 English publications, out of which 615k are open access. Many publications are annotated by experts with a variable number of subject headings (so-called subject indexing) taken from a standardized set, the "Thesaurus for Economics" (STW)[2]. We obtained an EconBiz dump from June 2017 provided by ZBW - Leibniz Information Centre for Economics[3]. We filtered the ones that are English and have STW annotations, and we extracted their title and STW labels. After deleting duplicates by comparing title and annotations, 1,064,634 publications remain. Out of the 615k open access publications, the number of publications that have annotations and whose full-text can be downloaded and processed reduces to 70,619, which is 6.63% of all publications.

PubMed[4] is a search engine for biomedical and life science literature provided by the US National Library of Medicine. The publications found on PubMed are annotated with "Medical Subject Headings" (MeSH)[5] by human curators. We obtained a dataset consisting of millions of publication metadata, including title and MeSH annotations, from the training set of the semantic indexing task of the BioASQ challenge 2017 [88], which are all in English language. PubMed Central[6] is an archive of full-texts of biomedical and life science literature provided by the US National Library of Medicine. It comprises 4.3 million publications, which can be accessed freely and which are mostly English. However, only 1.5 million are open access and therefore allow text mining. From this dataset, we computed the intersection with the publications obtained from the BioASQ challenge. After removing duplicates (again, by checking the title and annotations), 12,834,026 titles and 646,513 full-texts with respective annotations remain. Hence, 5.04% of the samples have a full-text.

[1]https://www.econbiz.de/
[2]http://zbw.eu/stw/version/latest/about
[3]http://www.zbw.eu
[4]https://www.ncbi.nlm.nih.gov/pubmed/
[5]https://www.nlm.nih.gov/mesh/meshhome.html
[6]https://www.ncbi.nlm.nih.gov/pmc/

Table 1 lists some characteristics of the two datasets, EconBiz and PubMed. In terms of combinatorial complexity, the PubMed dataset is a harder problem because the number of labels out of which to pick the annotations for a publication is much higher. Yet, due to the relatively large number of labels and small number of samples per label on average, both datasets can be considered as extreme multi-labeling classification problems[7]. The titles in PubMed contain on average more words than the publications' titles in EconBiz. However, this fact is put into perspective considering that the titles in PubMed have on average more labels to be predicted than there are words in the title. Regarding the full-texts, the ratio of words/labels is approximately the same in both datasets. Another fact worth noting is that the titles corpora have on average one label less than the full-texts. This can be explained by the fact that by design the full-text datasets comprise only open access publications. These tend to come from more recent years (compare Section C.1). This suggests that the label distributions in the title dataset and full-text dataset are quite different.

Please note that on both datasets, the set of titles is a superset of the set of full-texts.

4.2 Experiments

In order to assess how the title-based methods behave as more and more titles are considered for training, we create sub-datasets of the title datasets by iteratively adding more data. This is illustrated in Figure 1.

For a fair comparison of title and full-text performance, the trained model must be evaluated on the same data. To this end, we split the set of publications where a full-text is available into ten folds, and perform a 10-fold cross-validation. In each iteration, nine folds are selected for training, and one is selected for testing. From this training set, we randomly select 20% for the validation set for early stopping and adjusting the threshold, as described in Section 3.1. These comprise the data used for our experiments on full-text, and will be abbreviated as EconBiz Full and PubMed Full, respectively.

[7]An overview of datasets commonly considered extreme multi-labeling classification can be found at http://manikvarma.org/downloads/XC/XMLRepository.html.

	EconBiz (STW)		PubMed (MeSH)	
	Title	Full-Text	Title	Full-Text
$\|D\|$	1,064,634	70,619	12,834,026	646,513
Size	78.8MB	6.27GB	1.32GB	20.06GB
$\|L\|$	5661	4849	27773	26276
d/l	819.1	75.8	5852.3	331.0
l/d	4.4	5.3	12.6	13.5
$\|V\|$	91,505	1,502,336	660,180	6,774,130
w/d	6.88	6694.4	9.6	2533.4

Table 1: Characteristics of EconBiz and PubMed datasets. $|D|$ denotes the sample size, $|L|$ denotes the number of labels used in the dataset, d/l is the average number of publications a label is assigned to. l/d is the average number of labels assigned to a publication. $|V|$ is the size of the vocabulary and w/d denotes the average number of words per document.

For the experiments on titles, the same publications from the 10-fold cross-validation are used for testing. However, the training set is iteratively extended with more samples from the titles dataset, so that the total number of training samples is always a power of two of the number of samples in the full-text experiment. In total, we conduct experiments on five title sub-datasets per domain: T1, T2, T4, T8, and T_{all}. Here, Tx means that x times as many title samples are used for training as there are full-text samples in the dataset. Lastly, T_{all} contains all title samples from the dataset.

We run each of the four classifiers Base-MLP, MLP, CNN, and LSTM on all of the sub-datasets. In total, we run 48 cross-validations. Following Galke et al. [24] who argue that the sample-based F_1-metric best reflects how subject indexers work, we use this metric to report the results. We also use this metric for early stopping and threshold adjustment on the validation set. Formally, the F_1-score of a single sample x is defined as

$$F_1(x) = \frac{2 \cdot P(x) \cdot R(x)}{P(x) + R(x)}.$$

R(x) and P(x) refer to recall and precision, respectively, and are defined as

$$R(x) = \frac{|\gamma(x) \cap y(x)|}{|y(x)|}$$

$$P(x) = \frac{|\gamma(x) \cap y(x)|}{|\gamma(x)|}.$$

In this formula, $y(x)$ denotes the set of labels assigned to sample x according to the ground truth, while $\gamma(x)$ denotes the set of labels assigned by the classifier. The final sample-based F_1-score results as the average F_1-score over all samples in the test set.

4.3 Choice of Hyperparameters and Training

Since there are a lot of tunable hyperparameters involved in deep learning, tuning multiple hyperparameters at the same time can be very expensive, especially when the datasets are very large. On the other hand, fixing hyperparameters across all datasets and models would not be a fair approach in our study because the datasets and model architectures are very different and therefore may require very different hyperparameter settings. As a compromise, we decided to tune the hyperparameters for full-texts and titles separately in an incremental fashion. Here, we tuned one hyperparameter at a time and selected the locally best solution for full-text and titles, respectively. The entire tuning process can be traced in detail in Section C. It is important to note that the parameters for titles were determined based on the performance on T_{all}, and were adopted for all other title sub-datasets. On the one hand, this alleviates a lot of the computational cost and allows to compare the performance between title sub-datasets. On the other hand, especially the performance of smaller sub-datasets might be suboptimal due to overfitting. This must be kept in mind for analysis.

In our experiments involving the MLP, we use a one-layer MLP with 2,000 units and dropout with a keep probability of 0.5 after the hidden layer for all experiments. Only for the experiments on the PubMed titles, we use a two-layer MLP with 1,000 units each, and apply no dropout. Instead, we use Batch Normalization after each hidden layer. In all cases, the initial learning rate for Adam is set to 0.001.

In the CNN experiments, we use $p = 3$ chunks and $n_b = 1,000$ units at the bottleneck layer [51] for both full-text experiments. On titles, we do not perform chunking ($p = 1$), and use a bottleneck layer size of $n_b = 500$. The size of the feature map is set to 400 in all experiments except for PubMed Full, where we use 100. The keep probability is set to 0.75 in all cases, and the initial learning rate is 0.001.

We use a single-layer LSTM for all experiments. For both datasets, we determined 1,536 to be the best size for the memory cell when using titles. 1,024 units and 512 are used for PubMed Full and EconBiz Full, respectively. The keep probability is set to 0.75 in all experiments except for PubMed on titles, where we set it to 0.5. The initial learning rate is 0.01 for EconBiz Full and 0.001 in all other cases. Training is done with backpropagation through time by unrolling the LSTM until the end of the sequence.

We adopt the preprocessing and tokenization procedure of Galke et. al [24]. For the LSTM and CNN, we use 300-dimensional pretrained word embeddings obtained from training GloVe [66] on Common Crawl with 840 billion tokens[8]. Out-of-vocabulary words are discarded. The maximum sequence length is limited to the first 250 words.

For implementation of our neural network models, we used the deep learning library TensorFlow[9] and integrated them within the multi-label classification framework "Quadflor"[10] published by Galke et al. [24]. All experiments are run either on an NVIDIA TITAN or on a TITAN Xp GPU which both have 12GB of RAM.

5 RESULTS

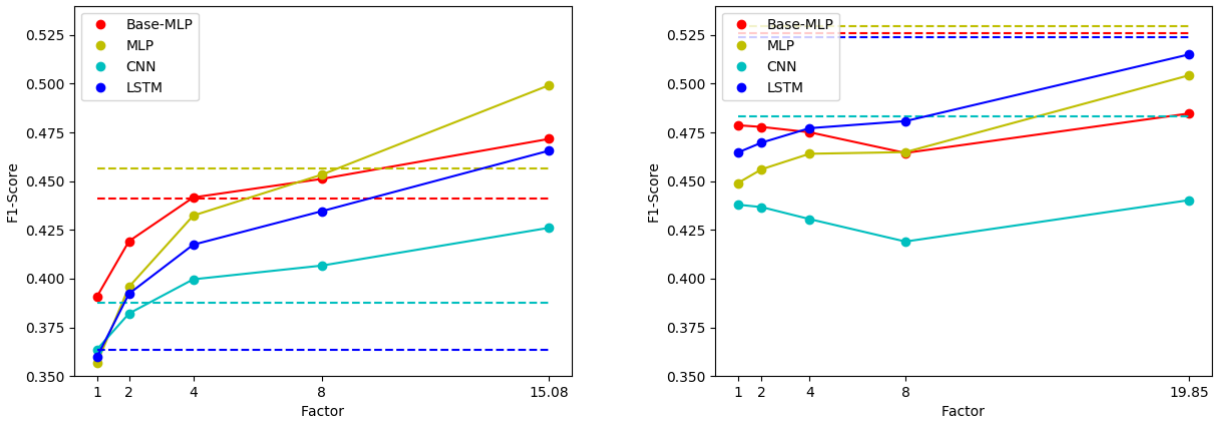
The results of our experiments are shown in Table 2. In addition, we plot the performance of each method as a function of the number of samples used for training the title model. These are shown in Figure 2.

EconBiz. On the EconBiz dataset, the best results on both titles and full-texts are obtained by MLP. The title-based method is on par

[8]This pretrained model can be downloaded at https://nlp.stanford.edu/projects/glove/.
[9]https://www.tensorflow.org/
[10]https://github.com/quadflor/Quadflor

Method \ Dataset	EconBiz F_1 scores						PubMed F_1 scores					
	Full-Text	T1	T2	T4	T8	T_{all}	Full-Text	T1	T2	T4	T8	T_{all}
Base-MLP	0.441	**0.391**	**0.419**	**0.442**	0.451	0.472	0.526	**0.479**	**0.478**	0.475	0.465	0.485
MLP	**0.457**	0.357	0.396	0.432	**0.453**	**0.500**	**0.530**	0.449	0.456	0.464	0.465	0.504
CNN	0.387	0.364	0.382	0.400	0.407	0.426	0.483	0.438	0.437	0.431	0.419	0.440
LSTM	0.363	0.360	0.392	0.417	0.435	0.466	0.524	0.465	0.470	**0.477**	**0.481**	**0.515**

Table 2: Results of experiments in terms of sample-based F_1-measure. The best performing method on each sub-dataset is printed in bold font.



(a) (b)

Figure 2: The figures show the performance of each classifier on titles as a function of the sample size relative to the number of full-texts as a solid line on EconBiz (a) and PubMed (b). The dashed horizontal lines represent the respective classifier's performance on the full-text.

with the full-text method when eight times as many titles as full-texts are used. When all titles are used, the title-MLP outperforms its full-text counterpart by 9.4%, achieving an F_1-score of 0.500. In contrast, when the same number of samples for full-text and titles is used, the gap between the best title method (Base-MLP) and the best full-text method (MLP) is 16.9% in favor of the full-text.

The MLP seems to benefit the most from additional titles. Initially, when as many titles as full-texts are used for training, all our proposed methods, i. e., MLP, CNN, and LSTM, perform within 0.007 points in F_1-score from each other, but MLP performs worst. However, the relation flips as more titles are added. In Figure 2 (a), we can observe that MLP has the steepest curve of improvement out of all methods, in particular when considering the improvement from T1 to T2. With twice as many titles as full-texts, MLP is already the best performing classifier out of the ones we have proposed. The gap to the other methods only gets wider as more data is added for training. Overall, the MLPs performance on T_{all} improves over the performance on T1 by 40.1%. The other methods also improve continuously as more training data is added. However, the CNN and LSTM improve by only 17% and 29.4% with respect to T1, respectively. Still, this is enough to surpass their full-texts counterparts by 10.1% and 28.4%, respectively.

When using as many titles as full-texts, Base-MLP outperforms our proposed methods clearly by 0.027 points in F_1-score. However, Base-MLP does not benefit as much from additional training data. Its overall improvement is only 20.7%, so when all titles are used for training, it is outperformed by MLP by a margin of 5.9%. On the full-text, MLP has an advantage of 3.6% over the baseline. Yet, the baseline still has a large advantage over LSTM and CNN, both on titles and full-texts.

PubMed. On PubMed, MLP shows the best full-text performance, whereas LSTM yields the best results on titles. However, even when all titles are used for training, that is almost 20 times as many titles as full-texts, the LSTM still lacks behind the full-text MLP by 2.9%. However, this is a considerably smaller gap than when the same number of samples are used. Base-MLP, the best performing method on PubMed T1, achieves 10.7% lower scores than the best method on PubMed Full, which is MLP.

The MLP and the LSTM show very similar behavior when more training data is added. This can be seen from Figure 2 (b), where the lines of the MLP and LSTM are almost parallel. However, the overall improvement from T1 to T_{all} is slightly higher for the MLP than for the LSTM. The former improves by 12.3%, whereas the latter

improves by 10.8%. The CNN does not seem to benefit from additional data at all. Initially, on T1, the CNN performs relatively close to the other methods, lacking behind the LSTM by 5.7%. However, as more data is added for training, the CNN demonstrates a worse classification performance than with fewer training samples. Only when all available titles are used, the CNN barely outperforms itself on T1 by a very slim margin of 0.5%. Consequently, the CNN has the largest difference to its full-text counterpart out of all proposed methods. It scores 9.8% lower on T_{all} than on PubMed Full, whereas the gap is 5.2% for the MLP and 1.8% for the LSTM.

By a considerable margin, the baseline is the best method on T1, where relatively few samples are used for training. However, despite using 20 times as many training samples, the performance on T_{all} is only 1.3% better, which is a difference to the MLP and LSTM of 3.9% and 6.2%, respectively. As it is the case on the EconBiz dataset, the full-text performance of BaseMLP is the second best and gets as close to the MLP as 0.8%.

6 DISCUSSION

The main question of our study is to which extent title-based methods can catch up to the performance of full-text-based methods by increasing the amount of title training data. On EconBiz, the best title-based method outperforms the best full-text method by 9.4% when all title training data is used. Considering that the difference is 16.9% in favor of the full-text when the sample sizes are equal, this is an impressive improvement. On PubMed, the improvement is less astounding. However, the title-based method is close to competitive to the full-texts, as the difference in score (less than 3%) is small. Considering that the gap is much larger for equal sample sizes (10.7%), we must acknowledge that current machine learning techniques in combination with large quantities of data are able to obtain just as good classification performance by merely using the titles.

However, in order to utilize title-based methods in a particular application, it is important to understand why there is such a large difference between the EconBiz and PubMed datasets regarding the benefit of employing title-based methods with large amounts of data. A possible explanation for that difference lies in the absolute numbers of full-texts available for training. As we have pointed out, previous literature suggests that deep learning models require around 650,000 samples to outperform more traditional approaches. On EconBiz, this number of full-texts is far from being reached. Due to this lack of enough training data, our deep learning models may not do so well with full-texts, in absolute numbers. The models based on titles on the other hand may be able to achieve their impressive results because there is just enough data to unleash the power of deep learning models. In fact, our MLP, which was optimized for large sample sizes, starts to outperform the baseline when eight times as many titles as full-texts are used, which nets to approximately 560,000 training samples. On the PubMed dataset, there are almost 650,000 full-text samples available. Here, the deep learning models can already work well on full-text. This may explain why the LSTM performs so much better on PubMed's full-texts than on EconBiz's full-texts. These findings support the claim from previous literature that deep learning models work well for text classification only when the sample size is several hundred thousands. Furthermore, since our datasets have large label spaces, we can state that this observation extents to extreme multi-label classification, which is arguably a harder task than single-label classification.

In order to push the limits of text classification based on titles, our strategy was to develop and employ methods that can make use of the vast amount of data available for training. Our results suggest that this strategy was largely successful. On both datasets, some of our methods surpass the performance of the baseline as they are given more and more data for training. BaseMLP on the other hand cannot make such good use of the additional training data. This becomes particularly apparent on the PubMed dataset, where it improves by only 1.3% even when it has 20 times more training data. Our methods on the other hand improve considerably the more data is used for training. To be fair, it is clear that part of the much larger gain compared to the baseline is due to overfitting on the small title datasets such as T1. Recall that our methods are optimized towards their performance on T_{all}. This design decision in our study was made to be able to fairly assess the development of the performance as the sample size increases. We observed that the capacity of the resulting models is likely too large for smaller datasets, which results in overfitting. This can be seen by the fact that BaseMLP, which has considerably lower capacity, outperforms our proposed methods on both PubMed and EconBiz. Yet, on T_{all}, MLP outperforms Base-MLP by a wide margin on both datasets. LSTM is close to Base-MLP on EconBiz and outperforms it drastically on PubMed. Again, this indicates the success of our strategy. The only exception is the CNN, which does not benefit as much from additional data as our other proposed methods, although its capacity is large compared to other architectures recently proposed in the literature. It is particularly interesting that the performance of CNN (and Base-MLP, too) actually has a drop on PubMed as more title samples are used for training. We explain this by the fact that in T2 to T8 more than 1,300 new labels that do not occur in T1 are introduced (for details please see Section C.1). Hence, the models have to account for these new labels even though they never appear in the test set, reducing their capacity to learn to classify the labels relevant to the test set.

Considering the amount of attention CNNs have received in recent years for their performance in text classification (cf. Section 2), the results of our CNN is rather underwhelming. This is true for both full-text, and titles. In neither case is this due to overfitting. Our preliminary experiments showed that the CNN benefits from increasing the feature map size a lot. Yet, the CNNs do not benefit from additional training data in the same way LSTMs and MLPs do. On PubMed, no benefit at all can be observed. On EconBiz, the rate of improvement is comparable to MLP and LSTM only up to a factor of four times the number of full-texts. After that, the improvement is relatively marginal. Moreover, we observed that after tuning the CNN its performance on the full-text improved by twice the relative amount than its performance on the titles (cf. Section D). In conclusion of our findings, we think it would be good if future research shifted its focus more towards MLPs and LSTMs, as they have demonstrated to be serious competitors for CNNs in text classification.

The goal of this study is to investigate to which extent a model trained on vast amounts of sample titles can compensate for the

lack of information in comparison to the full-text. Thus, it is not the aim to achieve results beyond the state-of-the-art performance on, e. g., full-texts. Yet, the models we use are based on and also enhance recently proposed models. As described in Section 3.1, the MLP is at its core a non-linear version of the popular fastText. The CNN is based on recent advances from the domain of extreme multi-label text classification, but was improved by integrating more fine-grained window sizes and larger feature maps. Finally, we present a strong bidirectional LSTM with attention over the outputs that does not assume a hierarchical structure of the document and therefore also works for short text snippets, in contrast to previous work by Yang et al. [93]. Therefore, our methods are good candidates for researchers to also adopt for single-label text classification.

A common problem in machine learning research and in deep learning in particular is that models are very sensitive to the choice of hyperparameters. However, examining the whole hyperparameter space is very difficult due to its combinatorial complexity. Recently, this has called the validity of deep learning results into question, for example in language-modeling [56] or even text classification [46]. This problem persists in our study as well. However, instead of simply assuming values for our parameters or manually tuning them in a somewhat arbitrary fashion, we took an incremental tuning approach that in the end led to an improvement over initial base models from the literature in all cases. This gives us reason to believe that our results are largely reliable. We discuss this further in the supplementary material in Section D.

In this study we have compared three deep learning methods. We did not compare against linear models such as logistic regression, and we did not compare against other non-linear approaches such as kNN or SVMs. However, as described in Section 2, there is evidence that traditional linear methods are inferior to non-linear ones when the training data is large. More importantly, we compared against a non-linear baseline by Galke et al. [24] that was shown to outperform not only linear models, but also other non-linear, non-parametric models like kNN and SVMs on a diverse set of datasets and by a wide margin.

For comparability, our proposed models were chosen such that they can be employed to titles and full-texts uniformly. This prohibits that certain strength of the full-text can be made full use of. For instance, full-text models might benefit greatly from a hierarchical model as proposed by Yang et al. [93]. On the other hand, we tried our best to tap the full potential of full-texts. For instance, in our CNN we employ dynamic max-pooling with $p = 3$ for full-texts although this does not have any beneficial effect on titles.

In our study, we have examined two datasets from digital libraries of scientific content. We argue that these results are likely to generalize to other datasets of scientific publications as well. In consent with previous text classification research, we found our deep learning methods to require approximately 550,000 samples to outperform the previous baseline (EconBiz T8). While this number of titles can certainly be reached in domains other than economics and biomedicine, not many scientific domains will reach this number of full-texts. This can be seen by considering the fact that the availability of full-texts is generally tied to their open access rate. The rate of open access journals in academia is approximately 7% as reported by Teplitskiy et al. [86]. This number closely matches the rate of 5 to 6.5% of available full-texts in our datasets. Moreover, Teplitskiy et al.'s study also suggests that the corpus of publications from the medical domain are among the largest. Therefore, we think it is likely that in other domains at least the relatively small gap of less than 3% between titles and full-texts can be achieved as well. However, in many other domains where only few full-texts are available, training models on titles may actually be much better, as we demonstrated for the economics domain.

Our results are of great practical importance for automatic semantic indexing in digital libraries. Considering the large amount of new literature published every year, human curators rely on the assistance of machines. It is desirable to have algorithms that produce good annotation suggestions by using only the title as textual input instead of the abstract or full-text. This is because the title is easy to obtain and free to use in text mining applications, whereas the full-text and even the abstract are often either not available or may not be processed automatically due to legal restrictions. For instance, in EconBiz only approximately 15% of the documents have an abstract and 7% have a full-text available. Furthermore, the task of downloading and processing the full-text is cumbersome, where one has tens of gigabytes of data. In contrast, as Table 1 shows, titles only comprise up to a few gigabytes of data. Thus, the full-text are by an order of magnitude larger. Thanks to the vast amount of data available for training title-based deep learning models, our study demonstrates that in a realistic scenario deep learning algorithms are able to satisfy the demand for sufficiently strong title-based classification methods. Engineers of (semi-)automatic semantic indexing algorithms should therefore consider shifting their focus from full-text-based classification to title-based classification in order to maximize the applicability of automatic semantic indexing systems.

7 CONCLUSION

In this paper, we have successfully answered the question if a semantic indexing system based on the title can reach the performance of a system based on the full-text if the number of samples for training the title-based method is much larger than the number of full-text samples. To this end, we developed three deep learning methods and evaluated them on two scientific datasets of different size. We found that in one case such a system is competitive with the full-text system, and in the other case it even yields considerably better scores. These results have important implications for automatic semantic indexing systems in digital libraries of scientific content.

In the future, we want to push the limits of classification based on large amounts of titles even further. To this end, we plan to adopt techniques that were out of scope for this study, but which we think also work particularly well with large sample sizes. These include leveraging information on label co-occurrence and integrating instance-level information into neural networks. Section E offers a more detailed description of our plans for the future.

Supplementary Materials

A EXTENDED RELATED WORK

In this section, we provide more detail on studies that are relevant to ours but not necessary to understand the main part of the paper. First, we discuss the research area of semantic subject indexing. Next, we review work that investigates the effect of the sample size on the classification performance. In Section A.3, we discuss further papers from deep learning research on text classification. Finally, we briefly present the research area of extreme multi-label classification.

A.1 Semantic Subject Indexing

Toepfer and Seifert [87] categorize subject indexing approaches into *statistical associative* and *lexical* approaches. The latter approach leverages domain knowledge provided with the thesauri to identify the correct label, e.g., by using keyphrase extraction [55, 59], graph-activation [30], or by learning label embeddings from a hierarchical thesaurus [62]. An advantage of such a lexical system is that it requires only several hundred training samples to work well [55]. However, as argued by Pouliquen et al. [68], this approach also requires a considerable amount of lexical resources such as rich thesauri and background information. In contrast, in the former approach, an algorithm learns to associate terms occurring in the document with labels assigned to the document without resorting to domain knowledge. This is typically approached as a multi-label classification problem, e.g., in [24, 53, 72]. As identified by Medelyan and Witten [55], these classification techniques require a relatively large amount of training data for every label, which is problematic in domains where the labels are distributed according to a power law. To overcome the shortcomings of the two approaches, Toepfer and Seifert [87] propose to combine them with label-invariant fusion techniques.

A.2 Effect of Training Sample Size

Formally, neural networks are a family of parametric statistical models. For learning based on the binary cross-entropy loss as in this study, the goal is to estimate the parameters Θ that best describe the conditional probability $P(y|x;\Theta)$, where y is the class to be predicted and x is the input. From the theory of statistics it is known that any consistent estimator of Θ approaches the true parameter values as the number of samples approaches infinity. Neural networks are trained with the consistent estimator "maximum likelihood". Hence, if all regularity assumptions are fulfilled (i.e., the data are sampled i.i.d. and the neural network can represent the true distribution), neural networks must eventually improve in performance as the number of samples increases.

Besides these theoretical results, some research has been concerned with empirical studies of the effect of the training set size on the classification performance. The results are mixed. Brain and Webb [12] observe that the overall error rate decreases as the sample size increases. Speaking in terms of bias-variance decomposition of the error, they find the variance decrease by a larger margin than the bias as the sample size increases, regardless of the applied estimator. They conclude that on large datasets the engineer should focus on learning algorithms which are able to reduce the bias. Since high bias corresponds to underfitting, the implications with respect to our study is to develop rather complex models such as in deep learning to adequately tackle very large title datasets. In other terms, we have focused on increasing the *capacity* of our algorithms. Emam and Elmaghraby [1] observed a logarithmic relation between the size of the training set and the performance on the test set when training a feed forward neural network. Therefore, their study supports the view that additional training data helps. On the contrary, Gatta et al. [26] found no significant impact of the training set size on the classification performance. Instead, the performance seemed to degrade when considering 2,000 training samples and more.

A.3 (Deep) Neural Networks for Text Classification

In Section 2, we have already discussed previous studies on text classification that use deep learning and that were most relevant to our study. Here, we provide more detail on some of those studies where necessary, and also discuss work not previously mentioned.

Yang et al. [93] proposed an RNN to learn a sentence representation from words and then use another RNN to learn a document representation from the sentence representations. The representation is obtained by taking the average over the RNN's output at each step. This hierarchical approach was evaluated on the large-scale text classification datasets introduced by Zhang et al [101] that have multiple sentences per text. On these, the method showed considerable improvement over the previous approaches. However, as Conneau et al [17] note, this method is not universally applicable since it requires multiple sentences per sample. Unfortunately, titles do not generally consist of multiple sentences (if any). Hence, in order to avoid a bias towards full-texts, we do not adopt the same hierarchical approach as Yang et al. [93] in this study. By employing a variant of the attention mechanism for averaging the outputs of the RNN, Yang et al. [93] are able to improve the results even more. Even though the effect of attention is not as large as the effect of the hierarchical document representation, we took inspiration and adopted their attention mechanism for our LSTM model.

Lenc and Král [48] conducted several studies on multi-label document classification, in which they use deep learning classifiers. Comparing a two-layer MLP that uses as features the information whether a word is present in the document with a CNN similar to that by Kim [42], they find the CNN to yield slightly better results in terms of F_1. Their neural network architectures are similar to the ones used in our study and the dataset used in the study consists of less than only 12,000 documents. Hence, it would be a valuable insight to know if these models can perform well on such a small dataset. However, the authors provide comparison with only one baseline from outside deep learning. Moreover, the dataset is in Czech language and therefore rarely used in other studies, making comparisons difficult. In a subsequent study, Lenc and Král conduct experiments on the more common Reuters-21578 dataset and employ ensembles of their neural networks [49]. However, even the best ensemble is not able to outperform the non-neural baseline on this dataset. This may indicate that also in multi-label classification neural networks require a very large number of samples in order to outperform traditional approaches.

Finally, Lenc and Král [50] explored a multi-labeling technique with strong similarities to the Neural MetaLabeler proposed in our study. The two methods were developed in parallel. Their method is based on the idea to use an MLP at the second level to predict the number of labels from the scores obtained from the neural network at the first level. However, in contrast to our approach, the two levels of their neural network are not trained jointly. Moreover, in contrast to Neural MetaLabeler, they do not present a counterpart to the content-based MetaLabeler approach (see Section B.3.2 for details).

A.4 Extreme Multi-label Classification

Extreme multi-label classification refers to classification on multi-label datasets with an "extremely" large number of labels. To the best of our knowledge, there is no widely accepted definition of how many labels are needed to be considered "extreme". The Extreme Classification Repository[11], a collection of datasets relevant to the research area, lists datasets that start at 101 labels. Hence, the datasets used in our study (see Table 1) can clearly be considered extreme classification. Common challenges in extreme classification are high computational demands and the fact that most of the labels occur only a few times because they follow a power law distribution.

Prior to deep learning, one of the most common approaches to deal with the large label space was to embed the labels within a dense, low-dimensional vector space (e.g., SLEEC [9]). After embedding the labels, the classifiers are trained as a regression problem to predict the label in the embedded space given the input. Another branch of research are tree-based approaches (e.g., FastXML [69]). Here, the assumption is that only a small number of labels is relevant in any particular region of the feature space. By partitioning the feature space into a tree structure whose branches can be traversed at prediction time according to the input, the number of labels to consider can be reduced considerably. Lastly, also the binary relevance classification scheme has been explored, where a separate classifier is trained for each label. This is usually not an option when dealing with a large label space because it is computationally too expensive and requires a lot of disk space to save the trained classifiers. However, these challenges can be alleviated by smart parallelization during training and by making the weight matrices of the resulting models sparse [3].

We are aware of two studies that have employed deep learning to tackle extreme multi-label classification. Zhang et al. [100] follow the embedding-based approach. They represent each label as a node in a graph and use the DeepWalk algorithm [67] to compute the nodes' embeddings. The feature space is embedded using a CNN. Liu et al. [51] propose to use a plain CNN directly for text classification, and their architecture is similar to Kim [42]. However, with a lot of possible labels the output layer becomes equally large. In order to keep the number of parameters introduced through the connections between the last hidden layer and the output layer acceptable, another narrow hidden layer (bottleneck layer) is added after the pooling stage. Please note that although the number of labels in our datasets is large enough to be considered extreme multi-label classification, it is still small enough to not cause problems in the number of parameters of the model. In none of our experiments with CNNs, the size of the model is limited by the available RAM of the GPU (12 GB). Nonetheless, we adopted the idea of the bottleneck layer because it introduces one more layer of abstraction to the CNN, and may therefore improve the generalization performance. Furthermore, we adopted from Liu et al. [51] the dynamic max-pooling technique, which is described in Section 3.3.

[11] http://manikvarma.org/downloads/XC/XMLRepository.html

B METHODOLOGICAL DETAILS

In this section, we provide formal details on the methods informally described in Section 3. Moreover, we provide details on methods that are used in experiments discussed in Section C. First, we discuss ways how a text can be transformed such that it can be processed by a neural network, whose architectures we discuss thereafter. In Section B.3, we frame the multi-label classification as a learning problem. Afterwards, we present the training procedure for that learning problem.

B.1 Text Representation

In any classification problem, representing a member of the population appropriately such that the classifier can work effectively is a difficult problem. For text in particular, the first challenge is to identify the atomic constituents of the text. In our case words, these are words, but it would also be possible to consider individual characters. The identification of individual words is done in the *tokenization* and *preprocessing* steps. In the subsequent step, two types of text representations are common in NLP-related tasks and are also made use of in this study. One of them, the *Bag-of-Words* (BoW) approach, disregards word order in the text but is in exchange able to capture information from the entire text. The other approach represents the text as a *sequence of tokens*, where each token is represented as a one-hot vector and is thus uniquely identifiable. In order to capture semantic similarities between words, these one-hot encodings can be mapped into a dense, low-dimensional vector space, so called *word embeddings*.

A key parameter is the dimensionality of the feature vector that represents the data. Obviously, the higher the dimensionality, the more information about the sample can be encoded as input and therefore be leveraged by the classifier, potentially improving its performance. However, the dimensionality also increases the complexity of the model in terms of tunable parameters, which has an impact on the computational complexity at both training and test time.

B.1.1 Tokenization and Preprocessing. For tokenization, we identify all non-overlapping sequences of alphabetic characters and the hyphen character. Subsequently, the hyphen is removed from all tokens. Finally, the token is lowercased and lemmatized based on WordNet [89].

Furthermore, when pretrained word embeddings are used (see Section B.1.3), we discard all tokens for which there is no entry in the lookup table of the pretrained model.

B.1.2 Bag-of-Words. In the BoW approach, a text d is represented as a feature vector x^d of the size of the *vocabulary* V. Component x_i^d denotes the score of word i in the text. The score is supposed to represent how important i is in this specific text d.

Intuitively, the more frequently i occurs in the text, the more important it is. Therefore, a common approach is to set $x_i^d = TF_d(i)$, where $TF_d(i)$ denotes the raw count how many times the word i appears in d. However, some words appear in many documents. Therefore, they do not carry much distinctive meaning for one particular text. An example is the word "the". The TF-IDF [78] retrieval model accounts for this by punishing the words depending on how often they appear in the entire corpus D. Formally, we set

$$x_i^d = TF_d(i) \cdot IDF(i, D),$$

where

$$IDF(i, D) = 1 + \log \frac{|D| + 1}{|\{d \in D : i \in d\}| + 1}.$$

A hyperparameter in the BoW approach is to select a suitable vocabulary. Clearly, the most informative representation is obtained by including every word that occurs at least once in the training corpus. In this case, however, two practical problems arise. First, the classifier has a harder time separating useful words from those that are not useful for prediction. Generally, words are useless if their feature representation has a small variance over all training samples, as is the case for words that appear very rarely. Secondly, given the size of the datasets used in our study, the vocabulary size is several million words, as Table 1 shows. This in turn increases the number of parameters, which can slow down training. Moreover, the maximum number of parameters is limited by the available RAM on the GPU that is used for training. The larger the vocabulary size and hence the amount of parameters needed to connect the features to subsequent layers of the network, the less capacity remains for the rest of the model, e.g., the number of units in a hidden layer.

Therefore, we need to limit the vocabulary in such a way that most of the information relevant for classification is still kept, but the size of the vocabulary allows for decent training speed. We determined a proper vocabulary experimentally, which is discussed in Section C.2.1.

B.1.3 Sequence of Tokens. CNNs and LSTMs consume the text as a sequence of tokens $s = t_1 \dots t_l$, where we assume each token to be in the vocabulary. Technically, both methods are able to cope with arbitrary sequence lengths. However, in order to enable fast training in batches, each sample has to have the same sequence length. Due to our large datasets, fast training is essential to leverage the information from the large amount of samples in reasonable time. Therefore, s must be padded or trimmed to always have the constant length n.

This proceeds as follows. If $l < n$, we append to the sequence as many zeros as are needed to reach length n, resulting in the modified sequence $t' = t_1 \dots t_l \vec{0}^{n-l}$. If $l > n$, we remove $l - n$ tokens from the end of the sequence. We argue that removing the tokens from the end is a reasonable approach in comparison to other options. For instance, one could remove the first $l - n$ tokens of the sequence. However, in classification of scientific documents, the beginning of a document is likely to contain the title and the abstract of the document, which are more indicative of the topic than the remainder of the full-text. Moreover, this procedure is in line with Zhang et al. [101], who also retain the first part of each text.

Word Embeddings. So far, we have considered the sequence $s = t_1 \dots t_l$ to be a list of strings. Obviously, neural networks cannot cope with string representations but require a numerical representation instead. Therefore, a token t_i is typically represented as a one-hot vector, in which all entries are zero except entry j, which is one. Here, j corresponds to the j-th word from the vocabulary, which equals token t_i.

However, a one-hot vector representation has disadvantages. First, similar to the BoW models, the feature representation can become very large if the size of the vocabulary is large. This is computationally inefficient. Second, one-hot vectors do not carry any semantic information about the words they represent. For instance, two semantically similar words have the same distance to each other in the BoW vector space as two semantically very different words.

To address these issues, one-hot vectors are usually transformed to k-dimensional, dense vectors, where k is very small compared to the vocabulary size. These representations carry semantic information in the sense that semantically similar words are mapped to vectors that are close in the k-dimensional space.

Algorithmically, the transformation from a one-hot vector to a dense vector is achieved through a simple linear transformation of the input. Let $x \in \{0, 1\}^{|V| \times 1}$ be the one-hot representation of a token, and let $W \in \mathbb{R}^{|V| \times k}$ be the so-called *lookup table*. The dense representation $x' \in \mathbb{R}^{k \times 1}$ is then obtained as the operation $x' = x^T W$. Since x is a one-hot vector with entry j set to one, this operation returns the j-th column from the lookup table.

The values in W can be learned jointly with other learnable parameters of the model during the training of the neural network by inserting the transformation as the first layer after the input layer. Alternatively, W can be loaded from a pretrained model that has resulted from an unsupervised training procedure such as word2vec [57] or GloVe [66]. The main advantage of pretraining word embeddings on unsupervised data is that their quality is not limited by the amount of labeled training data. On the other hand, the pretraining is unaware of the prediction task for which the word embeddings are intended to be used subsequently. Therefore, these unsupervised approaches have no incentive to incorporate information that are specifically helpful for the task. As a compromise, the lookup table may be initialized with pretrained word embeddings that are afterwards finetuned during training of the actual classification task.

In our experiments we deal with datasets of drastically different sizes. While the large datasets may be big enough to learn meaningful word embeddings, pretrained embeddings may be more expressive on smaller datasets. Therefore, we compare jointly learned embeddings with a number of pretrained models motivated from the literature. We also evaluate if finetuning of the pretrained models yields even better results for our datasets. For the above reasons, we evaluate this on each dataset separately.

B.2 Neural Network Architectures

In this section, we describe formally the neural networks with which we experimented in this study. We describe all architectures up to the last hidden layer, after which the output of the neural network is computed uniformly, which we describe now.

Let $x_h \in \mathbb{R}^{k_h}$ be the values after said last hidden layer of some neural network. We would like the model to output a probability estimate $\tilde{y}(x;\Theta) = (\tilde{y}_1, \ldots, \tilde{y}_{|L|})^T$ for each of the $|L|$ labels that the label should be assigned. Θ denotes the entirety of all learnable parameters in the model, such as weight matrices and biases. We obtain probabilities by applying the sigmoid activation function

$$\tilde{y}(x;\Theta) := \sigma(z_h) = \frac{1}{1+e^{-z_h}}. \tag{1}$$

Since its range is (0, 1), the output can be interpreted as the probability that a label should be assigned. z_h denotes an affine transformation of x_h,

$$z_h = W_o x_h + b_o,$$

where $W_o \in \mathbb{R}^{L \times k_h}$ and $b_o \in \mathbb{R}^L$ are the weight matrix and bias of the output layer, respectively.

B.2.1 Multi-Layer Perceptron. The multi-layer perceptron (MLP) is the most classical type of neural network architecture. It is a form of feed-forward neural network where there is a connection from each node in one layer to every node in the successive layer. Formally, let $1, \ldots, P$ denote the hidden layers of the network, n_l the number of nodes in layer l, and x_l the input to layer $l \in \{1, \ldots, P\}$. Here, we assume layer 1 be the input layer and layer P be the last hidden layer, hence $x_1 = x$. Mathematically, the connections between layers are implemented as an affine transformation $z_l = W_l x_l + b_l$, where $W_l \in \mathbb{R}^{n_{l+1} \times n_l}, b_l \in \mathbb{R}^{n_{l+1}}$. Subsequently, a so-called *activation function* f_l is applied, which constitutes the input at the next layer, so that $x_{l+1} = f_l(z_l)$.

In this study, we mainly use the rectifier $f(x) = \max(0, x)$ as activation function, which has established as a default activation function for neural networks [60].

When neural networks are very deep, they are prone to suffering from the exploding or vanishing gradient problem (compare [27, chapter 8.2.5]). The vanishing gradient problem refers to the effect that in neural networks that have many layers the gradient at the front layers may become very small, effectively leading to the weights not being updated at those layers. This may happen when the activations at the layers yield values smaller than one. Since the gradients are computed using the chain rule, these small values are multiplied, leading to even smaller values. The exploding gradient is caused by the same effect, but with the difference that the activations have initially large values. In order to mitigate these problems, an approach that has been followed repeatedly is to normalize the activations such that they are distributed with zero mean and variance one. Several techniques have been proposed to achieve this [2, 39, 44, 77]. In order to experiment with MLPs deeper than one hidden layer, we decided to employ Batch Normalization [39] and Self-Normalizing Neural Networks [44]. We decided to experiment with the former because it is the most commonly used normalization technique, and we experiment with the latter because it was only recently introduced, and has shown promising results for MLPs on a large number of datasets.

Dropout. Since its introduction, dropout [82] has become one of the standard techniques to prevent overfitting in large neural networks. The intuitive idea is to force the neural network to learn more robust intermediate representations of the input by randomly preventing the activations of a unit to propagate to subsequent units during training. This is achieved by setting the output of a neuron to zero with some probability $q = 1-p$, where p is called the *keep probability*. Formally, let $x \in \mathbb{R}^{n_l}$ be the input from some layer l. During training, we draw a sample $r_j \sim Bernoulli(p), 1 \leq j \leq n_l$ from a Bernoulli distribution. By multiplying this random sample with the original input, $x' = x \odot r$, where $\odot$ denotes the element-wise multiplication, we obtain the desired effect.

However, at prediction time, the input should not be distorted, i.e., we do not want to set units to zero at random. Instead, the units values are scaled by the keep probability, so that $x' = xp$. The reason is to ensure that the expected output of each neuron is the same at training time and at test time.

Self-Normalizing Neural Network (SNN). In order to tackle the vanishing and exploding gradient problem, Klambauer et al. [44] propose to normalize the output at each layer to zero mean and a variance of one using a specialized activation function that, as the authors proved, pushes the distribution of activation values to zero mean and variance one. That activation function is the "scaled exponential linear unit" (SELU), which is given by

$$selu(x) = \begin{cases} \lambda x & x > 0 \\ \lambda(\alpha e^x - \alpha) & x \leq 0 \end{cases},$$

where $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$. Moreover, the initial weights W_l from the affine transformation that is used at layer l have to be drawn randomly from $\mathcal{N}(0, \sqrt{\frac{1}{n_{l-1}}})$. Finally, Klambauer et al. [44] note that the dropout default value (zero in case of the dropout version described in the previous paragraph) should be set to the negative saturation value of the activation function, which in fact is zero for the commonly used rectifier activation function. For SELUs, however, the negative saturation value is $\lim_{x \to \infty} selu(x) = -\lambda\alpha$. Therefore, SNNs are used in combination with "alpha-dropout", which sets the value of a unit to $-\lambda\alpha$ with probability $1 - p$.

Batch Normalization. Batch Normalization [39] is another way to deal with the exploding and vanishing gradient problem. Here the idea is to normalize the distribution *within each mini-batch* $\mathcal{B} = \{x^1, \ldots, x^m\}, x^i \in \mathbb{R}^{n_{l-1}}$ of size m to zero mean and variance one. This is achieved through the following updating scheme:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x^i \tag{2}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu_{\mathcal{B}}) \tag{3}$$

$$\overline{x^i} = \frac{x^i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{4}$$

$$\overline{\overline{x^i}} = \gamma \odot \overline{x^i} + \beta \tag{5}$$

In these equations, (2) computes the mean over the mini-batch, (3) computes the variance, and (4) normalizes each sample in the batch, where ϵ is a constant added for numerical stability. Since the normalization step limits the representational power of the layer, (5) is introduced as an additional step to revert these limitations. Here, $\gamma, \beta \in \mathbb{R}^{n_{l-1}}$ are learnable parameters that are adjusted jointly with the other parameters in the neural network during training.

This mini-batch-based normalization scheme is only necessary and desired during training. At inference time, the normalization is performed with constant values that are computed in a preliminary step as the mean and variance estimates $E[x], Var[x]$ of the entire dataset. The normalized values $\overline{x^i}$ result as follows from the unnormalized values x^i:

$$\overline{x^i} = \frac{\gamma \odot x^i}{\sqrt{Var[x] + \epsilon}} + (\beta - \frac{\gamma E[x]}{\sqrt{Var[x] + \epsilon}}).$$

B.2.2 Convolutional Neural Network. Like MLPs, convolutional neural networks also belong to the family of feed-forward neural networks. However, in contrast to MLPs, CNNs are sparsely connected. Also, in contrast to the MLP, we consider the variable size word sequence $w_1, \dots, w_n$ as input. Let $x_1, \dots, x_n \in \mathbb{R}^k$ denote the corresponding sequence of word embeddings, obtained from a lookup-table as described in Section B.1.3. We denote the concatenation of the word embeddings as $x_{1:n}$, and the subsequence from word i to word j is analogously denoted as $x_{i:j} \in \mathbb{R}^{(j-i+1)k}$.

Base-CNN. Our initial model, which we later refer to as Base-CNN, is based on the architecture used by Kim [42].

At each position $i = 1, \dots, n - h + 1$ of the text, a CNN applies m filters to attempt to extract features $c_i \in \mathbb{R}^m$ from a window of h words. The feature extraction is implemented as the computation

$$c_i = f(W x_{i:i+h-1} + b), \tag{6}$$

where $W \in \mathbb{R}^{m \times hk}$ and $b \in \mathbb{R}^m$. The function f is called the *detector*. In this study, we again use the rectifier activation function (see Section B.2.1).

After applying the filters at every position, we are left with a sequence of length q,

$$\overline{c} = [c_1, \dots, c_q],$$

of detected features. Here, $q = n - h + 1$. There is a practical and a methodological reason to reduce this sequence. The practical one is that the final hidden variable of any architecture must be of fixed size in order to compute the values at the output layer as explained in Section B.2. The methodological one is that in text classification it is usually not very important where in the text a certain key-phrase appears. It is rather critical to detect its mere absence or presence. Both reasonings are reflected in the max-pooling aggregation technique. In max-pooling, we look for the position in the input where a feature detected by a filter has the strongest signal, and extract only the value at that position:

$$MaxPool(\overline{c}) = (\max_{i=1,\dots,q} c_{i,1}, \dots, \max_{i=1,\dots,q} c_{i,m})^T. \tag{7}$$

The hyperparameters here are how to choose m, the number of filters (or also referred to as "feature map size", and h, the size of the window. Specifically, Kim [42] employs $m = 100$, but chooses multiple values $h = 3, 4, 5$ at once. The outputs of all filters of different window sizes are concatenated after max-pooling to constitute the final hidden layer. For our Base-CNN, we use these hyperparameter values, too.

Dynamic Max-Pooling. Liu et al. [51] argue that standard max-pooling is problematic on long texts because it only extracts one value for each feature from the entire text. Therefore, we adopt their idea to extract the maximum from different parts of the text. To this end, we split the output from the detector stage $\overline{c}$ (Equation B.2.2) into p chunks of approximately equal size $s = \lceil \frac{q}{p} \rceil$. Instead of applying max-pooling globally, it is applied on each chunk individually:

$$DynMaxPool = [MaxPool(\overline{c}_{i \cdot s:\min(q,(i+1)\cdot s-1)})]_{0 \le i \le p-1}$$

Bottleneck Layer. Liu et al. [51] propose to inject a bottleneck layer between the pooling stage and the output layer. A bottleneck layer is simply a fully-connected layer as is used in MLPs (compare Section B.2.1).

B.2.3 Long Short-Term Memory. The Long Short-Term Memory network (LSTM) belongs to the family of recurrent neural networks (RNNs) [22]. RNNs consume the input sequentially. Hence, we again consider the sequence of k-dimensional word embeddings $x_1, \dots, x_n \in \mathbb{R}^k$ obtained from the lookup-table. The sequence is consumed one input at a time. We refer to the current input as x_t.

Intuitively, an LSTM maintains a cell state c_t of size m that encodes information about the sequence up to step t. Moreover, an LSTM is able to control the flow of information through an input, a forget, and an output gate. Whether the gate is open or closed is determined by taking into consideration the output y_{t-1} from the previous step, the cell state c_{t-1}, and the current input x_t. Formally, the first step is as follows:

$$z_t = \tanh(W_z x_t + R_z y_{t-1} + b_z), \tag{8}$$

$$i_t = \sigma(W_i x_t + R_i y_{t-1} + p_i \odot c_{t-1} + b_i), \tag{9}$$

$$f_t = \sigma(W_f x_t + R_f y_{t-1} + p_f \odot c_{t-1} + b_f). \tag{10}$$

$W_z, W_i, W_f \in \mathbb{R}^{m \times k}, R_z, R_i, R_f \in \mathbb{R}^{m \times m}$ and $b_z, b_i, b_f, p_i, p_f \in \mathbb{R}^m$ are learnable parameters. Equation 8 yields a non-linear transformation of the input. Equation 9 and Equation 10 show the computation of the input and forget gate, respectively. These information can now be used to update the cell state:

$$c_t = z_t \odot i_t + c_{t-1} \odot f_t.$$

Given the updated cell state, we may now compute the value of the output gate (Equation 11) and the output, which is a non-linear activation of the cell state (Equation 12):

$$o_t = \sigma(W_o x_t + R_o y_{t-1} + p_o \odot c_t + b_o), \tag{11}$$

$$y_t = \tanh(c_t) \odot o_t. \tag{12}$$

where again $W_o \in \mathbb{R}^{m \times k}$, $R_o \in \mathbb{R}^{m \times m}$, and $b_o, p_o \in \mathbb{R}^m$.

For convenience, we denote the output as a result of one step in the LSTM as $y_t = LSTM(x_t)$. Throughout the supplementary material, we sometimes refer to this model as Base-LSTM. Following Zhang et al. [101], we set $m = 512$ and average the outputs.

Bidirectional LSTMs. In bidirectional LSTMs [80], the input is read both from left to right and from right to left. To this end, we make use of two LSTMs. The first LSTM was introduced above and produces outputs $y_1^f, \dots, y_n^f$. The second LSTM reads the sequence backwards, producing outputs $y_n^b, \dots, y_1^b$. The combined output from both LSTM is now a simple concatenation of the two outputs:

$$y_t = [y_t^f; y_t^b]$$

Stacked LSTMs. Multiple LSTMs can be stacked on top of each other in order to allow for more abstract representations of the input. When P layers of LSTMs, $LSTM_1, \ldots, LSTM_P$ are stacked, the input sequence $x_1, \ldots, x_n$ becomes the input to $LSTM_1$. In intermediate layers, the input x_t^i of $LSTM_i$, $1 < i < P$, is the output from the previous layer $LSTM_{i-1}$:

$$x_t^i = LSTM_{i-1}(x_t^{i-1}).$$

Consequently, the final output $y_1, \ldots, y_n$ results as the output of $LSTM_P$.

Aggregating the Outputs. After consuming the entire input sequence, the (potentially stacked) LSTM has produced n outputs $y_1, \ldots, y_n$. However, in order to propagate the output to the next layer, these outputs have to be aggregated to yield a fixed size vector. Common choices for aggregation are considering only the last output from the sequence, or computing the sum over all outputs, or computing the average over all outputs. Besides these three options, we also consider a weighted average, where the weights are obtained through the attention mechanism.

We adopt the "word attention" computation scheme from Yang et al. [93] to obtain the weights. First, a non-linear activation is computed for each output y_t:

$$u_t = \tanh(W_w y_t + b_w).$$

Again, $W_w \in \mathbb{R}^{\tilde{k} \times m}, b_w \in \mathbb{R}^{\tilde{k}}$ are learnable parameters. $\tilde{k}$ is a hyperparameter that controls the size of the context vector. Following Yang et al. [93], we set $\tilde{k} = 100$. The weights are computed by combining the non-linear activation with a context vector $u_w \in \mathbb{R}^{\tilde{k}}$ and using the softmax activation to take all other outputs into account. Formally, we compute

$$\alpha_t = \frac{\exp(u_t^T u_w)}{\sum_{t'=1}^{n} \exp(u_{t'}^T u_w)}.$$

The final step is to compute the weighted average of the inputs as

$$y = \sum_{t=1}^{n} \alpha_t y_t.$$

OverEager LSTMs. During the development of our models, we experimented with a simple trick, which we call *OverEager LSTMs* (OE-LSTMs). The trick does the following. As explained in Section B.1.3, it is necessary to limit the sequence to a maximum length in order to train the neural network efficiently. In the case where the sequence is shorter than maximum length, OE-LSTMs replicate a special padding token until the end of the sequence. Subsequently, they regard the sequence as any other sequence, that is, they iterate until the maximum length and also produce outputs at time steps where the padding token is consumed. We call this type of LSTM "OverEager" because it does more than it is usually supposed to do. Subsequently, the extra outputs are also considered at the aggregation step. Thus, they contribute to the final output, which is propagated to the next layer.

B.3 Framing the Multi-Label Classification Problem

As described in Section 3.1, the neural network makes a binary decision whether to assign a label by checking if the predicted probability exceeds a threshold θ.

In this section, we describe how the model can learn to make better predictions by defining the training loss objective and turning it into an optimization problem. Moreover, we propose an alternative way to obtain hard binary decisions via an adaptation of the MetaLabeler technique.

B.3.1 Training Objective. Our objective function is to minimize binary cross-entropy given the sigmoidal outputs of the final layer. Formally, let $y = \{0, 1\}^{|L|}$ be the indicator vector representing the ground truth such that $y_i = 1$ if label i is assigned to the sample x, and $y_i = 0$ otherwise. Recall that $\tilde{y}(x; \Theta)$ denotes the predicted probability determined by the model which is parameterized over Θ. The binary cross-entropy loss of x, $BCE(x)$, is computed as follows:

$$BCE(x) = -\sum_{i=1}^{L} y_i \log \tilde{y}(x; \Theta)_i + (1 - y_i) \log(1 - \tilde{y}(x; \Theta)_i). \quad (13)$$

The total loss of a batch is computed as the average over all n samples in the batch,

$$BCE_{batch} = \frac{1}{n} \sum_{x \in batch} BCE(x).$$

Finally, learning is framed as an optimization problem by parameterizing the prediction function $\tilde{y}$ over the set of parameters Θ. We look for the optimal parameters $\tilde{\Theta}$ that yield the smallest loss:

$$\tilde{\Theta} = \min_{\Theta} BCE_{batch}.$$

In order to find the minimum, we employ the gradient-based method Adam (see Section B.4.1). The gradients are computed with backpropagation [76].

Please note that our true objective is not to optimize the parameters with respect to a single batch, but with respect to the entire dataset. Because this converges rather slowly, a single optimization step is instead performed on a batch of samples as described here (compare Ruder [74]). By changing the batch in each iteration, the true objective is approximated.

B.3.2 Neural MetaLabeler. Meta-Labeler is a multi-labeling strategy proposed by Tang et al. [84]. In the following, we introduce the ideas of the algorithm and describe how we adapt them for neural networks, which we call *Neural MetaLabeler* (NML).

Concept of MetaLabeler. MetaLabeler proceeds in two phases. In phase i), a model predicts scores for assigning each label. In phase ii), another model predicts how many labels should be assigned. In the end, the predicted number of labels with the highest scores are assigned.

Phase i) can be implemented in a straight-forward fashion by any classifier that can provide confidence scores s_l for every label l. In our case, the confidence scores will be given as the output of the sigmoid layer of the neural network (Equation 1).

For phase ii), the authors propose several approaches. Let y_{ML} be the true number of labels defined as a multi-class classification

problem. Further, let $x_{ML} = \phi(x)$ be the input to some classification algorithm c_{ML} that tries to predict y_{ML}, produced by applying some function ϕ to the original input x. The proposed approaches differ in how ϕ is defined. Using $\phi_c(x) := x$, the prediction of the number of labels is performed directly on the content of the sample. $\phi_s(x) := [s_{l_1}, \ldots, s_{l_L}]$ assumes that it is sufficient to only look at the scores for each label as produced in phase i). Finally, the third option is to provide c_{ML} with a vector of sorted scores: $\phi_r := \Phi([s_{l_1}, \ldots, s_{l_L}])$, where Φ sorts the scores $s_{l_1}, \ldots, s_{l_L}$. Note that the difference between ϕ_s and ϕ_r is that in the former c_{ML} may learn label-specific score dependencies, while in the latter it can learn to associate the rank with the scores, i.e., identify gaps among the top scores more easily.

Tang et al. [84] evaluated their approaches on two text categorization datasets. For both phase i) and ii), a One-vs-Rest SVM is used as classifier. On the first dataset, a web categorization dataset, ϕ_c has a slight advantage over ϕ_s, whereas ϕ_r performs considerably worse. On the second dataset, the popular news dataset RCV1, ϕ_c clearly outperforms both alternatives.

We have reason to believe that the MetaLabeler strategy may be very powerful in our use case. For extremely large datasets, as is the case in our title datasets, phase ii) of the MetaLabeler may yield particularly good results since there is an equally large amount of examples for c_{ML} to learn from. Hence, it is a reasonable assumption that this multi-labeling variant benefits titles much more than full-texts.

Adaptation of MetaLabeler for Neural Networks. Since the study by Tang et al. [84] has shown that the ranking-based approach does not have a great effect, we focus on incorporating a combination of ϕ_c and ϕ_s in our neural networks.

We achieve this by learning to predict the total number of labels to assign *jointly* with learning to predict the concrete labels to assign. The general framework for such an approach is called *multi-task learning*. For a recent overview of multi-task learning in neural networks, please refer to the work by Ruder [75].

Obviously, the idea of multi-task learning is to learn to solve multiple tasks on the same input data at the same time. Typically, one task is considered the main task that we would like to solve optimally, and the other tasks are regarded as auxiliary tasks, whose purpose is to help to solve the main task better. In our case of multi-label classification, predicting the probabilities for each label is the main task, while predicting the number of labels is the auxiliary task. The motivation is that the auxiliary task introduces an inductive bias that makes the neural network prefer hypotheses that can explain more than one task. Again, in our case, we would like the model to prefer hypotheses that also explain the number of labels correctly.

Specifically, we employ a version of *hard parameter sharing* where both tasks share the complete model up to the output layer. Let x_h denote the output of the last hidden layer of the model as explained in Section B.2. Note that this is some function of the input x (the content). Hence, our adaptation of the content-based MetaLabeler is simply $\phi_c(x) = x_h$. For the score-based MetaLabeler, we resort to the predicted probabilities: $\phi_s(x) = \tilde{y}(x)$. Note that the prediction of the number of labels directly depends on x_h in the former case, and indirectly depends on it in the latter case. Hence, the parameters used to obtain x_h are shared among both tasks.

In the same fashion as Tang et al. [84], we frame prediction of the number of labels as a multi-class classification problem of classes $c_{\min}, \ldots, c_{\max}$ where $c_{\min}$ and $c_{\max}$ denote the minimal and maximal number of labels to be predicted, respectively. For simplicity, we chose $c_{\min} = 1$ and set $c_{\max}$ to the maximum number of labels assigned to any sample in the training set. In similar fashion as for the prediction of labels, we obtain probability estimates $\tilde{y}^c(x; \Theta')$ by applying a non-linear activation function with range (0,1). However, in contrast to predicting the concrete labels where we employ sigmoid, here we use $softmax$ to obtain the probability for class j:

$$\tilde{y}^c(x; \Theta')_j = \frac{e^{z'_{h,j}}}{\sum_{i=\min}^{\max} e^{z'_{h,i}}}.$$

Here, z'_h is an affine transformation of $\phi(x)$.

For optimization, again the binary cross entropy loss is used, denoted as $BCE'(x)$. In order to jointly optimize the multi-label objective and the MetaLabeler objective, the final objective function results as a linear combination of the two, controlled via the hyperparameter $\gamma \in (0, 1)$:

$$BCE^{NML}_{batch} := \frac{1}{n} \sum_{x \in batch} (1 - \gamma) BCE(x) + \gamma BCE'(x).$$

Intuitively, γ allows to control the degree of trading off between the two objectives. The higher the chosen value for γ, the more emphasis is put on correctly predicting the number of labels.

B.4 Training Procedure

Training neural networks is considered to be more difficult than training other machine learning algorithms. This can be explained by several factors. Neural networks have a lot of expressive power, especially when the number of units is large. Therefore, they are more prone to overfitting because they can easily approximate any target function [37]. Furthermore, the optimization objective when training neural networks is highly non-convex. Hence, an optimizer can easily get stuck in a local minimum that does not necessarily correspond to a good global solution.

Therefore, it is important to apply both a smart optimizer and a validation procedure that assesses the generalizability of the current solution. To address the former issue, we employ the well-known *Adam* optimizer. For the latter issue, we perform *early stopping on a validation set.*

B.4.1 Adam. Adam has established as a default optimizer choice since its introduction in 2014 by Kingma et al. [43]. It is a member of the family of adaptive optimization algorithms, which adjust the learning rate for each parameter. A non-adaptive strategy would be Stochastic Gradient Descent (SGD). Adam has shown to outperform or match the performance of other adaptive strategies, hence, it is seen as the best choice among them. Furthermore, Adam is said to work well with default hyperparameter values [74], which make it favorable if the number of external hyperparameters to tune is large. Recent research by Wilson et al. [91] challenges this view and suggests superiority of SGD in terms of generalizability when the hyperparameters are carefully tuned. Since we examine

a wide range of models on large datasets, we decide to avoid the combinatorial explosion that would arise if we had to tune the hyperparameters of the optimizer. Therefore, in our experiments, we use Adam with its default parameters suggested in the original paper [43] ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). We only tune the initial learning rate α.

B.4.2 Early Stopping on Validation Set. Early stopping is a technique that allows to monitor the model's performance as it is trained. To this end, a fraction r of the training set, called the validation set, is withheld from training. After processing $\tilde{b}$ batches (or weight updates), the model's classification performance on the validation set is assessed. The parameters of the weight update after which the validation score is highest are saved and loaded at test time. The training is stopped after p consecutive epochs with no improvement over the best model. p is commonly referred to as *patience*.

Since the performance on the validation set approximates the generalization performance, this is an effective technique to avoid overfitting. The larger r and consequently the validation set, the closer does the validation performance approximate the true generalization performance. On the downside, the fraction r is not available for training, which ultimately hurts the model's capacity. Nonetheless, early stopping is recommended to be used universally in every application [27, p. 425]. Therefore, we apply this technique in our experiments.

We choose $r = 0.2$, which is a commonly recommended rule how to split the training data [27, p. 121]. There does not exist a universal rule how to set the patience p. While it is tempting to think that the validation error is a convex function of the number of weight updates that has a single minimum at which to terminate training ($p = 0$), Prechelt [70] argues that the function is non-convex and has many local minima. In such cases higher patience may be required to find the global minimum. Unfortunately, in Prechelt's examination, the stopping criteria are not robust, but the optimal criterion depends on the learning task instead. Therefore, we search for a choice of p in our preliminary experiments (see Section C.2.2). Moreover, we also determine a reasonable value for $\tilde{b}$ in our preliminary experiments.

B.4.3 Hyperparameters of Training Procedure. In addition to the hyperparameters we have already discussed or set to a fixed value, there are two more hyperparameters of the training procedure that require a separate discussion: the batch size and the threshold θ for assigning a label.

Batch Size. Mini-batch training is an important technique to speed up the training of neural networks. It refers to the process of propagating multiple training samples at once through the network, during forward propagation as well as backpropagation. GPUs then allow to do the propagation efficiently in parallel. Therefore, with respect to the computational efficiency, it is advisable to work with large batch sizes. However, there are a few limitations to the size of the batches. First, the batch has to fit into the GPU memory. How many samples per batch we can have therefore depends on the number of input features. Moreover, since the model's parameters (the weights) have to be kept in GPU memory as well to update them efficiently, the size of the model further limits the batch size. Finally, the generalization error tends to increase if the batch size is too large. Keskar et al. [41] have analyzed this so-called "generalization gap" and have shown empirically that relatively small batches of size 256 indeed yield better test performance than very large ones that consist of 10% of the training data. A few solutions to this problem have been proposed. Byrd et al. [13] as well as Keskar et al. [41] propose a dynamic batch size, which is small at the earlier stages of the training and is increased as the training progresses. This procedure supposedly decreases the chances of converging to a *sharp minimum*, which allegedly causes the generalization gap. However, recent work by Dinh et al. [19] disagrees with this view. As an alternative explanation, Hoffer et al. [35] argue that the generalization gap when training with large batches is due to a smaller number of weight updates. The discussion around this phenomenon shows that it is not well understood yet. Therefore, in order to avoid the generalization gap altogether, we set the batch size to 256 and accept slower training speeds.

Threshold. As described in Section 3.1, we optimize the threshold θ by assessing the F_1-score on the validation set after each epoch. At best, we would like to determine

$$\theta^* = \underset{\theta \in [0,1]}{\arg\max} F_1(D_{val}; c, \theta), \tag{14}$$

where $F_1(D_{val}; c, \theta)$ computes the F_1-score after applying the cutoff threshold θ on the probabilities estimated by classifier c for samples in the validation set D_{val}. However, it is very costly to perform an exhaustive search over the entire space $[0, 1]$ with sufficiently small steps.

We argue that the F_1-score is almost a concave function of the threshold. This is due to the fact that the average precision monotonically decreases as θ increases, whereas the average recall monotonically increases in θ. Hence, by raising θ, we trade precision for recall, and we trade recall for precision by lowering θ. While it is not generally true, we may heuristically assume that the sum of precision and recall remains approximately constant. Consequently, for any given fixed sum $S \overset{!}{=} P + R$, the maximal F_1-score for choices of precision P and recall R is achieved when $P = R$. This can be seen by observing that the F_1 score boils down to the equation

$$F_1 = \frac{2 \cdot P \cdot R}{S},$$

whose local maximum under the constraint that $P + R = S$ is at $P = R = \frac{S}{2}$.

From these observations we derive the following heuristic. Before training, we choose an initial threshold value $\theta := \theta_0$. In each validation step i, we set

$$\theta_i := \underset{\theta \in \{-k*\alpha+\theta_{i-1}, \ldots, k*\alpha+\theta_{i-1}\}}{\arg\max} F_1(D_{val}; c_i, \theta), \tag{15}$$

where c_i denotes the classifier with the weights of the model after $i \cdot \tilde{b}$ weight updates, $\alpha > 0$ is the step size and $k \in \mathbb{N}$ is a parameter that controls the number of threshold values to check in each validation step ($2k + 1$ in total).

C INCREMENTAL ALGORITHM DESIGN

The success of neural networks at some specific task heavily depends on the choice of architecture and hyperparameters. In order to assure a fair comparison between the models, it is important

to select the hyperparameters such that neither of the models is favored solely because of the particular choice of hyperparameters. Unfortunately, examining the whole hyperparameter space (*grid-search*) is computationally very expensive, and particularly infeasible if training a single configuration already takes a long time due to very large datasets.

In order to reduce the computational complexity of grid-search, some other automatic search methods for hyperparameter optimization have been proposed. Bergstra et al. [7] introduced *Random Search*, which randomly draws a configuration to evaluate from the hyperparameter space. In their study, their experiments showed comparable results to grid-search, but needs to evaluation much fewer configurations. Snoek et al. [81] have employed Gaussian processes for Bayesian Optimization. Instead of making a blind guess as in Random Search, in Bayesian Optimization approaches, an algorithm estimates the next configuration to evaluate from the performance of previously evaluated configurations. Their approach shows superiority over other popular hyperparameter estimation methods like Random Search and the Tree Parzen Estimator [8] on several datasets.

Generally, hyperparameter optimization is not an easy task. There exist no methods which can efficiently estimate optimal scores. Hence, optimization is often performed manually, relying on human expertise. However, human expertise is naturally inconsistent and therefore inappropriate for a fair comparison of models. Moreover, it is a very intransparent approach, which is a reason why hyperparameter optimization is often seen as “more of an art than a science" [27, p. 295].

Therefore, in preliminary experiments, we experimented with both Random Search and Bayesian Optimization. The hyperparameters to optimize were the learning rate and the keep probability in dropout. In order to keep the overall computational time at an acceptable level, we limited the number of configurations to evaluate to 10. However, neither of the two automatic hyperparameter optimization methods was able to yield results that were close to the quality of setting the parameters manually with an educated guess (i.e., hyperparameters that performed well in related studies). The results obtained from Bayesian Optimization were still unsatisfactory when it was provided with three relatively good starting configurations.

The rather poor performance of the automatic hyperparameter search methods can be explained with the low number of evaluations. For instance, in the study on Random Search by Bergstra et al. [7], Random Search needs 32 evaluations or more to achieve good results with neural networks on the basic MNIST dataset. Clearly, we would also get better results with more trials. However, as mentioned before, even a single evaluation already takes a long time.

Therefore, we take an incremental approach of tuning the hyperparameters and the neural network architectures. Some hyperparameters can be viewed as largely independend of the architecture, the dataset, and the other hyperparameters, and in particular the learning rate. Examples are the pretrained word embeddings and the vocabulary size. For other hyperparameters, however, it may be necessary to optimize them for each type of neural network and dataset individually, such as the learning rate and dropout. Therefore, we evaluate nine different combinations of these two hyperparameters on all datasets and neural networks individually.

Again, for a fair comparison, it would be ideal to repeat the evaluation of different combinations of the learning rate and the keep probability for each change in the architecture of a neural network. In fact, Bengio [5, p. 8] suggests that the learning rate “is often the single most important hyperparameter and one should always make sure that it has been tuned”. However, this would effectively increase the required computation time by at least 3, which is too costly for our study given the large datasets and the fact that a lot of the architecture changes we consider have hyperparameters themselves. To cope with this, we have to assume the learning rate and keep probability to be relatively insensitive to architecture changes, and keep them fixed across the experiments regarding design choices in neural network architecture.

All experiments in this section are evaluated on one fold (commonly referred to as the development set) to limit the computational complexity during model development.

The rest of this section is organized as follows. Subsequently, we provide more detail on the datasets used in our experiments. We then present the preliminary experiments, after which we discuss how to fix the hyperparameters. In Section C.4, we perform a grid-search for the learning rate and dropout. We present experiments that are concerned with architectural design decisions specific to a neural network in Section C.5. We present experiments where we increase the capacity of the neural networks in Section C.6, before we lastly conduct some experiments with Neural MetaLabeler.

C.1 Datasets

In this section, we provide some further detail on the datasets used in this study beyond what is already described in Section 4.1. We describe in detail how the full-text datasets were obtained. Moreover, we discuss some more statistics to support statements made earlier in the paper.

Compilation of EconBiz Full. The EconBiz full-text dataset was crawled by a member of ZBW - Information Centre for Economics in August 2016. To this end, an EconBiz dump from the previous month was filtered for entries where the open access flag is set. Furthermore, the dump contains URIs to the PDF of the full-text for some of the open access entries. These URIs were automatically accessed and the corresponding PDFs were downloaded. Where this was successful, the PDF was converted to a Unicode file using the conversion library Apache PDFBox[12]. The resulting full-text dataset was merged with a dataset from 2014, which had been obtained through the same process. By intersecting with the EconBiz title dataset, only the English full-texts that have annotations remain.

Compilation of PubMed Full. PubMed Central provides an FTP service[13] for downloading the full-texts of the open access publications, where the full-texts can be downloaded either as a bulk containing raw text files or as a bulk containing XML files. Unfortunately, in the former bulk, the naming of the file containing a full-text is not by the publication’s id, which makes it difficult to identify to which publication a full-text belongs. In the XML bulk,

[12] https://pdfbox.apache.org/
[13] ftp://ftp.ncbi.nlm.nih.gov/pub/pmc/oa_bulk/

the naming is by id. Therefore, we downloaded the XML bulks and converted the XML files to text files in the following way. We traversed the XML tree to find the XML nodes labeled "article-title", "abstract", and "body". Where these were available, we extracted their content. The title was always available, and in all cases either the body (≈ 98%) or the abstract (≈ 99.9%) was available. Because the abstract and body could contain XML tags themselves, we used the ElementTree XML Python API to remove these. Finally, the extracted content of the three tags "article-title", "abstract", and "body" were concatenated using a newline as separator, and written to a file.

Number of labels by sub-dataset. In Section 6, we explain the performance drop of the CNN on PubMed T8 with a large number of new labels that are introduced in larger sub-datasets. This is shown in Table 3. While the relative gain is larger on EconBiz than on PubMed, the absolute number of new labels introduced between T2 and T8 is more than 2.5 times larger on PubMed than on EconBiz.

	EconBiz			PubMed		
	|L|	rel. gain	abs. gain	|L|	rel. gain	abs. gain
T1	4,849	-	-	26,267	-	-
T2	5,165	6.5%	316	27,135	3.3%	868
T4	5,230	1.3%	65	27,447	1.1%	312
T8	5,357	2.4%	127	27,626	0.7%	179
T_{all}	5,661	5.7%	304	27,773	0.5%	147

Table 3: Number of different labels in EconBiz and PubMed sub-datasets. Relative gain shows the percentaged increase in number of labels compared to the next smaller sub-dataset. Analogously, absolute gain shows the increase in absolute values.

Distribution of years by sub-dataset. As discussed in Section 4.1, Table 1 shows that the full-text samples are assigned approximately one label more on average. Figure 3 contrasts for each dataset the distribution of publication years in the sub-dataset T_{all} with the distribution of years in the sub-dataset Full. On both datasets, we can observe that almost all of the full-text samples were published later than the year 1995, whereas a large portion of the title samples were published in earlier years. This disparity is not surprising since the full-text datasets were compiled from open access publications only. The rise of the open access movement is closely tied to wide spread access to the internet, which developed in the 1990s.

The difference in the average number of assigned labels per sample may be explained by changes in the way human annotators work that may have developed over time. Moreover, since the popularity of research topics changes over time, and also new research topics emerge, the full-texts and titles datasets are likely very different.

C.2 Preliminary Experiments

We perform a number of preliminary experiments in order to determine good hyperparameter values that we can fix for subsequent experiments. We call these experiments "preliminary" for different reasons. Either the classifier is not a neural network (vocabulary size), or the dataset is not EconBiz or PubMed (early stopping and threshold learning), or the performance of the hyperparameter is largely independent of learning rate and dropout (word embeddings, sequence length).

Unless specified otherwise, in the subsequent experiments, MLP, CNN, and LSTM refer to the base models, Base-MLP, Base-CNN, and Base-LSTM. CNNs and LSTMs use an embedding size of 300 without pretraining. Following Galke et al [24], we set the learning rate to $\alpha = 0.01$ and the keep probability to $p = 0.5$. For the validation frequency $\tilde{b}$ we choose one epoch, and we set the patience to 20. The binary decision whether or not to assign a label is made with the threshold method, where the threshold is adjusted during training. For MLPs, the input is a TF-IDF vector with the 50,000 most common words in the vocabulary.

C.2.1 Vocabulary size. On each (sub)-dataset, we would like to find a small vocabulary for BoW-based classifiers (i.e., MLPs), such that little to no information relevant to the prediction task is lost. We achieve this by performing preliminary experiments where we show that the classification performance does not suffer from restricting the vocabulary to the m words that appear most frequently in the training corpus. In particular, we assess the impact of the choice of m on the classification with the *kNN* algorithm ($k = 1$). At test time, the algorithm assigns the same labels that are assigned to the closest sample in the training set, where distance is measured in terms of cosine similarity. By choosing kNN as our classifier, we adopt the reasoning by Galke et al. [24] that the kNN algorithm is suitable for assessing the quality of the features. For kNN in particular, a good feature representation is critical to achieve a good classification performance.

In the way formally described in Section B.1.2, we compute a TF-IDF representation of the text, but we only retain the m most common terms. We run one experiment per dataset on each of T1, T_{all}, and Full. This is necessary because these sub-datasets have very different vocabularies due to different domains and sample sizes. We run the 1NN algorithm on a 90:10 training-test split.

Please consider the results in Table 4. On the EconBiz dataset, the differences between scores on the same sub-dataset are 0.003 at maximum across all vocabulary sizes. On the PubMed dataset, the differences are larger, but in inconsistent directions. While the performance on titles decreases by up to 0.013 when going from unlimited vocabulary size to 25,000, the performance on full-texts increases by 0.005 when going from unlimited to 25,000.

C.2.2 Early Stopping and Threshold Learning. Subsequently, we describe three preliminary experiments that are concerned with the early stopping procedure and threshold learning. First, we assess the effect of our procedure by comparing to the results from Galke et al. [24]. Afterwards, we determine reasonable values for the hyperparameters $\tilde{b}$ and the patience p.

Effect of Early Stopping and Threshold Learning. We assess the effect of early stopping and the threshold optimization approach introduced in Section B.4.3 by evaluating on the datasets used in Galke et al. [24] (Economics, Political Science, Reuters, NYT - please see Galke et al [24] for details). Our implementation of Base-MLP is the same as the MLP architecture from Galke et al. [24], however, there are a few differences in the training procedure. First, Galke et

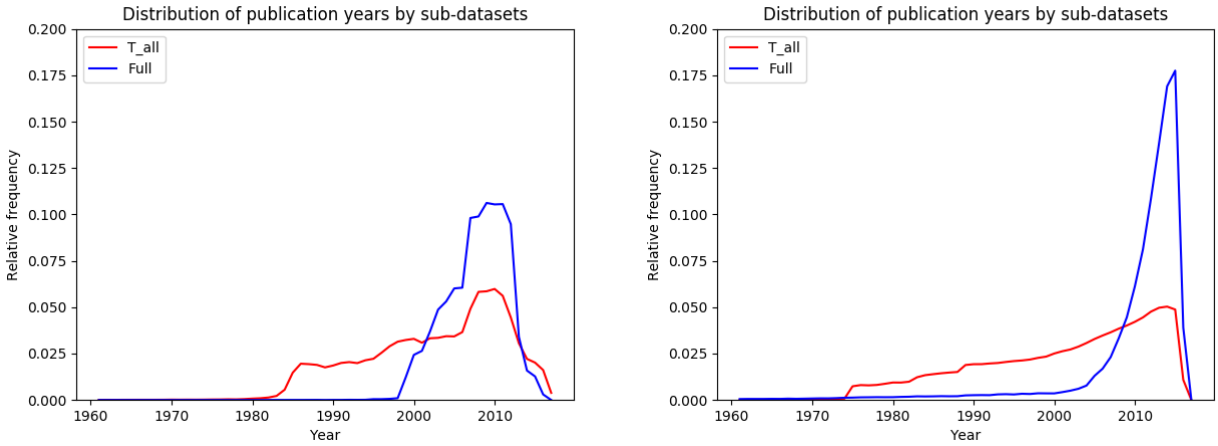


Figure 3: The distribution of publication years (1960 - 2018) in the T_{all} and Full sub-datasets on EconBiz (a) and PubMed (b).

Dataset	Sub-Dataset	∞	100,000	50,000	25,000
EconBiz	T1	**0.271**	**0.271**	**0.271**	**0.271**
EconBiz	T_{all}	0.394	0.394	**0.395**	**0.395**
EconBiz	F	**0.320**	**0.320**	0.319	0.317
PubMed	T1	**0.293**	0.291	0.289	0.283
PubMed	T_{all}	**0.316**	0.313	0.310	0.303
PubMed	F	0.320	0.321	0.323	**0.325**

Table 4: Results for 1NN with different experimental setups on both PubMed and EconBiz. ∞ denotes that there is no limit in vocabulary size.

al. [24] use a fixed threshold $\theta = 0.2$ throughout all experiments. Second, they do not perform early stopping on a validation set, but train the MLP for a fixed number of 20 epochs instead. Therefore, we conduct two additional experiments per dataset: By *MLP∗*, we denote Galke et al.'s [24] architecture but with early stopping on a validation set. By Base-MLP, we denote the variant where the threshold is additionally optimized on the validation set during training.

For comparability, we use the same hyperparameters as Galke et al. [24]. We perform a full 10-fold cross-validation in each experiment. For threshold optimization, we choose $k = 3$ and $\alpha = 0.01$ as a compromise between speed and granularity.

For the results, please consider Table 5. First, we look at the behavior when performing early stopping on a validation set as opposed to not doing that (MLP vs MLP*). The effects vary a lot across the datasets. On full-texts, the performance drops considerably on all datasets but Reuters, where we can observe a marginal increase of 0.003. Most notably, the difference for NYT is as much as 0.083. On titles, however, early stopping helps for Reuters and NYT, and the decrease on the economics and political science datasets are smaller than on full-texts. Second, we can observe that additionally optimizing the threshold on the validation set (MLP* vs Base-MLP) always helps. While the difference is rather small on Reuters (0.004), learning the threshold boosts the performance by up to 0.017 on the other datasets. Nonetheless, in 4 out of 8 cases, the performance with early stopping and threshold learning is considerably worse than when no early stopping is used (MLP vs Base-MLP).

Text	Classifier	Economics	Political	Reuters	NYT
Full-text	MLP	**0.519**	**0.373**	0.857	**0.569**
Full-text	MLP*	0.482	0.348	0.860	0.486
Full-text	Base-MLP	0.496	0.361	**0.864**	0.496
Titles	MLP	**0.472**	0.309	0.812	0.332
Titles	MLP*	0.446	0.291	0.819	0.345
Titles	Base-MLP	0.453	**0.314**	**0.823**	**0.362**

Table 5: Comparison of threshold methods. "MLP" refers to a fixed threshold $\theta = 0.2$, "MLP*" refers to the same fixed threshold, but early stopping is used. "Base-MLP" refers to the variant where the threshold is optimized during training. Scores for "MLP" stem from [24].

Patience. As discussed in Section B.4.2, determining a good patience value p is crucial to find a good trade-off between reducing the chance to get stuck in a sub-optimal local minimum that yields poor generalization performance, and acceptable training time. To this end, we examine for each type of neural network and each dataset (PubMed and EconBiz) the maximum number of consecutive validation steps during which the validation performance does

not improve over the previous best, but does improve eventually. We train each network for 200 epochs. We assume that this behavior only depends on the nature of the dataset and classifier, but not on the number of samples used for training. Therefore, we only evaluate on the T1 sub-datasets.

The results are shown in Table 6. While the MLP never improves again once it has not improved after a validation step, the CNN and LSTM still eventually improve even after multiple consecutive validations of no improvement (up to 9).

Dataset	Model	Max validations
EconBiz	Base-MLP	0
EconBiz	CNN	5
EconBiz	LSTM	1
PubMed	Base-MLP	0
PubMed	CNN	3
PubMed	LSTM	9

Table 6: Maximum number of consecutive validation steps without improvement.

Validation Frequency. When running our neural networks with a validation frequency $\tilde{b}$ of one epoch (a commonly chosen standard value), we noticed that on PubMed the performance of the titles declined when adding more training data. In our experiments, T1, T2, and T4 achieved an F_1 score of 0.462, 0.452, and 0.440, respectively. A reason could be that due to the size of the PubMed dataset, the classifier reaches its optimal parameter values in terms of validation set performance somewhere in the middle of an epoch, but has already overfit the dataset at the end of that same epoch. As the validation set performance is only assessed at the end of each epoch, it uses an overfitted model at test time and thus yields bad performance. In order to verify that this is indeed what causes the drop in performance when more training samples are added, we conducted experiments where we determine the validation set performance after a set number of batches, i.e., weight updates.

The larger the dataset, the more likely is the above scenario of overfitting to happen. Therefore, we only looked at our largest datasets, i.e., T_{all} from EconBiz and PubMed, respectively. We assess the validation performance after 2,000, 1,000, and 500 batches, respectively, and compare to when the validation is performed after an entire epoch. Since the illustrated problem could occur irrespectively of the type of neural network, we only run experiments with the MLP.

The results in Table 7 show the F_1-scores of Base-MLP. On EconBiz, the differences are barely noticeable. On PubMed, there too is no difference between validation after 500 batches, 1,000 batches, and 2,000 batches, respectively. However, when validating after a whole epoch (after seeing 25× as many training samples before validation compared to 500 batches), the performance drops drastically by 0.094 in F_1-score.

Dataset	Validation Frequency	F_1- score
EconBiz	1 epoch	**0.446**
EconBiz	2,000 batches	0.443
EconBiz	1,000 batches	0.445
EconBiz	500 batches	0.443
PubMed	1 epoch	0.355
PubMed	2,000 batches	0.448
PubMed	1,000 batches	**0.449**
PubMed	500 batches	**0.449**

Table 7: F_1-score of Base-MLP when validating after a full epoch, or 2,000, 1,000, and 500 weight updates, respectively.

C.2.3 Word Embeddings. In Section B.1.3, we discussed the importance of (pretrained) word embeddings. Several methods to train word embeddings from unlabeled data have been proposed in the past. Here, we experiment with GloVe [66][14] and word2vec [57][15], which have shown to be the most successful on intrinsic tasks [79] and are commonly used. Additionally, we experiment with embeddings that incorporate subword information (called *fastText* here), which have recently been shown to improve the quality of embeddings further [10][16].

Due to differences in the vocabularies they were trained on, different pretrained embedding models may perform differently depending on the dataset. We therefore evaluate them on PubMed and EconBiz separately. In contrast, the quality of embeddings may be assumed to benefit both CNNs and LSTMs by the same relative amount. Therefore, for a relative comparison of the embeddings, we run experiments only with LSTMs and will assume that CNNs will perform accordingly. Moreover, we assess whether finetuning the pretrained word embeddings helps the classification performance or not.

The results are shown in Table 8. An immediate observation is that the effect of finetuning is very inconsistent across the datasets. While it benefits fastText on PubMed by 0.024 in terms of F_1-measure, it degrades the performance on EconBiz by 0.043. GloVe shows slight improvement on both datasets with finetuning, while word2vec is always weakened. It is noteworthy that even with no pretrained embeddings the model can perform better than with some pretrained embeddings that are finetuned. Overall, finetuned fastText wins on PubMed, but is closely followed by finetuned GloVe. On EconBiz, GloVe is the winner.

C.2.4 Sequence Length. As described in Section B.1.3, in order to enable training in batches, we need to limit the length of the sequence. The titles datasets have a maximum sequence length of 60 (PubMed) and 84 (EconBiz), respectively, which is not problematic. Therefore, we conduct experiments only on the full-text datasets. Moreover, since CNNs and LSTMs react very differently to the length of the sequence, we need to assess its impact on both

[14]This model can be downloaded at https://nlp.stanford.edu/projects/glove/.
[15]This model can be downloaded at https://github.com/mmihaltz/word2vec-GoogleNews-vectors.
[16]This model can be downloaded at https://fasttext.cc/docs/en/english-vectors.html (wiki-news-300d-1M-subword.vec.zip).

Dataset	Embeddings	Raw	Finetuned
EconBiz	None	-	0.356
EconBiz	word2vec	0.359	0.324
EconBiz	GloVe	0.371	**0.372**
EconBiz	fastText	0.367	0.324
PubMed	None	-	0.420
PubMed	word2vec	0.430	0.418
PubMed	GloVe	0.430	0.442
PubMed	fastText	0.421	**0.445**

Table 8: Comparison of pretrained word embeddings with and without finetuning. None denotes that embeddings are randomly initialized.

classifiers. Finally, the effect of the sequence length might also differ depending on the dataset. This can be seen when considering that full-texts in PubMed are on average much shorter than full-texts on EconBiz (see Table 1). In our experiments, we evaluate sequence lengths 100, 250, 500, and 1,000, respectively.

The results in Table 9 show that the top performance on both datasets and with both models is reached at a sequence length of 250. However, the CNN is rather invariant with respect to the lengths, which results in relatively small differences in F_1-score of 0.025 at maximum. The LSTM, on the other hand, is very sensitive to the length of the sequence. On EconBiz, it benefits a lot from seeing more words up to 250, increasing the performance by 0.066 compared to a sequence length of 100. The performance is stable when increasing the length to 500, but collapses completely when increasing the length to 1,000. Unfortunately, we can not provide results for length 1,000 on PubMed because the model does not fit into the 12 GB RAM of the GPU.

Dataset	Sequence Length	LSTM	CNN
EconBiz	100	0.281	0.261
EconBiz	250	**0.347**	**0.284**
EconBiz	500	0.341	0.277
EconBiz	1,000	0.053	0.267
PubMed	100	0.470	0.333
PubMed	250	**0.489**	**0.335**
PubMed	500	0.458	0.331
PubMed	1,000	-	-

Table 9: Comparison of different values for the maximum sequence length in full-texts.

C.3 Discussion of Preliminary Experiments

In this section, we discuss the results from the preliminary experiments. Most importantly, we determine how to fix the hyperparameters for subsequent experiments.

C.3.1 Vocabulary Size. The results from applying 1NN have shown that on EconBiz there is no considerable difference even when reducing the vocabulary size to 25,000. On PubMed, the differences are larger but inconsistent throughout experiments. On the other hand, due to the considerations on memory requirements made in Section B.1.2, we argue that it is reasonable to accept a possibly small decrease in performance due to a smaller vocabulary if in turn the capacity of the neural networks can be increased. Therefore, in subsequent experiments with MLP we restrict the vocabulary size to 25,000.

C.3.2 Early Stopping and Threshold Learning. Subsequently, we discuss the results from the preliminary experiments regarding early stopping and threshold learning.

Effect of Early Stopping and Threshold Learning. The experiments have shown that in many cases the early stopping procedure yields worse results than training for a fixed number of epochs. This can be explained by the fact that 20 percent of the training data must be used as a validation set when early stopping is used, and are thus not available for training. On datasets where the performance increases nonetheless, the regularizing effect of early stopping is able to compensate for the shortage of training data. For example, on the NYT title dataset, we observed the maximum performance on the validation set to be achieved after 5 epochs already and the optimal threshold was at 0.07. Here, the number of 20 training epochs and $\theta = 0.2$ manually chosen by Galke et al. [24] are too high. However, the same parameters work very well on all full-text datasets. It is unclear where this disparity between titles and full-texts come from. An explanation could be that for full-texts the input features are not as sparse as for titles, which requires the MLP to make more weight updates (i.e., training needs to proceed for more epochs). For the same sparseness argument, additional training data, which is not available when using early stopping, may be crucial for fitting the parameters.

When early stopping is already used, the experiments clearly show that it is always beneficial to also optimize the threshold on the validation set during training. Hence, it is only debatable whether early stopping (and thus sacrificing training data) should be used in the first place.

We argue that it should be used for a number of reasons. First, for a fair comparison among datasets and classification methods, we need to aim for independence from the particular hyperparameter settings. Therefore, it is advisable to optimize as many hyperparameters as possible locally, which our proposed training procedure does for the number of weight updates and the threshold. Secondly, a manual optimization would be too costly on datasets as large as the ones considered in our study. Lastly, the drop in performance with early stopping is due to a reduction in the training sample size. Since the classification performance is approximately a logarithmic function in the number of training samples [1], the drop due to less training data will be considerably larger on relatively small datasets. In their experiments, Galke et al. [24] use datasets with

up to only 100,000 samples. In our study, however, the majority of (sub-)datasets is much larger. In summary, we decided to use early stopping and threshold learning in subsequent experiments.

Patience. The experimental results suggest that the MLP has found the global optimum in terms of validation set performance once the validation performance decreases. This gives reason to assume that the MLP is relatively easy to optimize. On the other hand, the LSTM and CNN seem to be more difficult to train. However, as described in Section B.4.2, this is not easy to determine, and in order to make a well-grounded judgment, further analysis would be required, such as a full plot of the validation performance over time. For our study, however, we are only interested in a reasonably good value for the patience. Since our experiments suggest that $p = 10$ is sufficient to not get stuck in a local optimum, we fix $p = 10$ for subsequent experiments.

Validation Frequency. The experimental results on PubMed T_{all} confirm our presumption that on such a large dataset the model both reaches its maximum validation performance and overfits within the same epoch. In fact, when running the experiments, we observed that this happened during the first epoch already.

Since a higher validation frequency did not make much difference for the EconBiz dataset but helped the PubMed dataset dramatically, we will set the validation frequency to $\tilde{b} = 2,000$ in subsequent experiments.

C.3.3 Word Embeddings. We will use GloVe with finetuning globally in our subsequent experiments. On EconBiz, our decision is justified because GloVe shows the best performance. On PubMed, fastText with finetuning is slightly better. However, during training with the fastText embeddings, we observed that the optimizer got stuck at a poor solution in terms of validation score performance of about 0.10, and remained there for up to 15 consecutive validations. In contrast, word2vec and GloVe did not show such a behavior during training. Considering the relatively small performance gain with fastText over GloVe, and the danger of fastText getting stuck at a solution far from the optimum, we argue that it is best to use GloVe also on PubMed.

C.3.4 Sequence Length. On EconBiz, both classifiers perform considerably worse when the sequence length is limited to 100, whereas on PubMed, this is not the case. This suggests that the first 100 words of an EconBiz document are not enough to cover its full topical breadth, whereas for PubMed it is enough. This interpretation is supported by the fact that PubMed documents are on average much shorter than documents from EconBiz (compare Table 1).

The LSTM seems to fail to learn the prediction task sufficiently when increasing the length from 500 to 1,000 on EconBiz. This may be explained by the vanishing or exploding gradient problem, which is particularly prominent in RNNs if the errors are backpropagated a large number of recurrent iterations, as shown by Bengio et al. [6]. In contrast, it is not clear why the performance of the CNN does not improve, but becomes slightly worse instead as the length of the sequence increases beyond 250. A possible explanation could be that the CNN becomes more prone to capturing noise from later parts of the document.

Our experimental results clearly suggest that 250 as maximum sequence length should be used for all subsequent experiments since it outperforms all other configurations for both models on both datasets.

C.4 Tuning the Learning Rate and Dropout

In the previous two sections, we determined values for hyperparameters that we assumed to be largely independent of the learning rate. In this section, we want to determine good values for the learning rate and dropout's keep probability.

C.4.1 Experimental Results. In order to determine a good combination of learning rate and dropout values, we performed a grid-search with each of Base-MLP, Base-LSTM, and Base-CNN. Because 0.5 is a recommended default value for a wide range of tasks [82], we chose values $p \in \{0.25, 0.5, 0.75\}$ to also evaluate cases where more or less regularization than usual is required, respectively. For the learning rate, Galke et al. [24] used $\alpha = 0.01$. We also include the values 0.1 and 0.001 in our grid-search, which are by an order of magnitude larger and smaller, respectively.

Subsequently, we present the results for the MLP, CNN, and LSTM separately.

MLP. The results of the grid-search for MLP are shown in Table 10. In the experiments with MLP, a learning rate of $\alpha = 0.001$ is superior in all cases. If any at all, dropout rates make only a small difference of up to 0.005 in F_1-score. A learning rate of $\alpha = 0.01$ still yields reasonably good results on both datasets, but do considerably worse in some cases, by up to 0.075 in F_1-score. Finally, setting $\alpha = 0.1$ makes the training diverge, yielding unsatisfactory results on both datasets.

CNN. Table 11 shows the result of the grid-search on CNNs. On both PubMed and EconBiz, $\alpha = 0.001, p = 0.75$ is the best hyperparameter configuration in all cases. Similar to the MLP, the performance degrades as the learning rate increases, again making the neural network fail to learn at $\alpha = 0.1$. In contrast to the MLP, the keep probability can make a large difference, resulting in up to 0.14 difference in F_1-score between a lot of regularization ($p = 0.25$) and little regularization ($p = 0.75$).

LSTM. Results for the LSTM are shown in Table 12. As opposed to CNN and MLP, no single best configuration can be determined. Instead, both $\alpha = 0.01$ and $\alpha = 0.001$ yield reasonably good results. It is inconsistent which of the learning rate values performs better depending on the sub-dataset, however, except for the T_{all} datasets, where $\alpha = 0.001$ is clearly better, the performances are rather close to each other. $\alpha = 0.1$ again shows poor performance and even runs into a numerical error in one case (marked as "-"). The keep probability makes a larger difference than with the MLP, but a smaller difference than with the CNN. On PubMed, $p = 0.75$ always performs best, whereas on EconBiz, all three values show best performance in at least one case.

C.4.2 Discussion of Results. The results are very clear for MLPs and CNNs and the choice of hyperparameters for subsequent experiments. The learning rate in particular appears to be a stable choice. For both classifiers, we will choose $\alpha = 0.001$ due to its superiority over the alternative values. For MLPs, the difference

Dataset		α	p	F_1-score	Dataset		α	p	F_1-score
EconBiz	Full	0.001	0.25	0.439	PubMed	Full	0.001	0.25	**0.527**
EconBiz	Full	0.001	0.50	0.439	PubMed	Full	0.001	0.50	**0.527**
EconBiz	Full	0.001	0.75	**0.442**	PubMed	Full	0.001	0.75	**0.527**
EconBiz	Full	0.010	0.25	0.424	PubMed	Full	0.010	0.25	0.472
EconBiz	Full	0.010	0.50	0.425	PubMed	Full	0.010	0.50	0.472
EconBiz	Full	0.010	0.75	0.429	PubMed	Full	0.010	0.75	0.470
EconBiz	Full	0.100	0.25	0.149	PubMed	Full	0.100	0.25	0.240
EconBiz	Full	0.100	0.50	0.146	PubMed	Full	0.100	0.50	0.239
EconBiz	Full	0.100	0.75	0.160	PubMed	Full	0.100	0.75	0.241
EconBiz	T1	0.001	0.25	0.393	PubMed	T1	0.001	0.25	**0.479**
EconBiz	T1	0.001	0.50	**0.396**	PubMed	T1	0.001	0.50	**0.479**
EconBiz	T1	0.001	0.75	0.394	PubMed	T1	0.001	0.75	**0.479**
EconBiz	T1	0.010	0.25	0.387	PubMed	T1	0.010	0.25	0.438
EconBiz	T1	0.010	0.50	0.393	PubMed	T1	0.010	0.50	0.438
EconBiz	T1	0.010	0.75	0.389	PubMed	T1	0.010	0.75	0.438
EconBiz	T1	0.100	0.25	0.139	PubMed	T1	0.100	0.25	0.220
EconBiz	T1	0.100	0.50	0.140	PubMed	T1	0.100	0.50	0.220
EconBiz	T1	0.100	0.75	0.138	PubMed	T1	0.100	0.75	0.223
EconBiz	T_{all}	0.001	0.25	**0.475**	PubMed	T_{all}	0.001	0.25	**0.486**
EconBiz	T_{all}	0.001	0.50	**0.475**	PubMed	T_{all}	0.001	0.50	**0.486**
EconBiz	T_{all}	0.001	0.75	**0.475**	PubMed	T_{all}	0.001	0.75	0.484
EconBiz	T_{all}	0.010	0.25	0.407	PubMed	T_{all}	0.010	0.25	0.416
EconBiz	T_{all}	0.010	0.50	0.407	PubMed	T_{all}	0.010	0.50	0.414
EconBiz	T_{all}	0.010	0.75	0.408	PubMed	T_{all}	0.010	0.75	0.415
EconBiz	T_{all}	0.100	0.25	0.109	PubMed	T_{all}	0.100	0.25	0.218
EconBiz	T_{all}	0.100	0.50	0.108	PubMed	T_{all}	0.100	0.50	0.217
EconBiz	T_{all}	0.100	0.75	0.108	PubMed	T_{all}	0.100	0.75	0.218

Table 10: Results of grid-search for MLP on EconBiz and PubMed. α denotes the learning rate, and p denotes the keep probability of dropout.

between dropout values is not large. We pick $p = 0.5$ in subsequent experiments because it has the largest margin to regions where training could become unstable ($p < 0.25, p > 0.75$) because we have no information about them. For CNNs, $p = 0.75$ clearly outperforms the alternatives.

On LSTMs, the results are less clear, showing rather large inconsistencies regarding the optimal learning rate and keep probability. This may hint at the difficulty to train LSTMs properly, and may cause problems when changing the parameters or the architecture of the LSTM. Unfortunately, as mentioned earlier, in this study we do not have the computational resources to repeat a grid search for every architectural design change. Therefore, we will use the parameters that have performed best here in subsequent experiments. On PubMed, we will use $\alpha = 0.001$ and $p = 0.75$. On EconBiz, we will use $\alpha = 0.01, p = 0.75$ on the full-text and $\alpha = 0.001, p = 0.5$ on titles.

C.5 Incremental Neural Network Architecture Design

In this section, we incrementally explore the effect of small changes to our neural networks as they were described in Section B.2. First, we discuss the performance of our base models in order to compare them with the final models. We then make incremental changes to the design of each type of neural network separately.

In all subsequent experiments, all hyperparameters are chosen according to the results discussed in Section C.3 and Section C.4.2. Moreover, we conduct experiments on the Full, T1, and T_{all} subdatasets.

C.5.1 Experiments with Base Models. For convenience, we list the results of the base models in Table 13. The scores correspond to the respective hyperparameter configuration that is also used in subsequent experiments. Hence, these values stem from Tables 10, 11, and 12.

Base-MLP outperforms the other methods by a wide margin on EconBiz. On PubMed, the LSTM is slightly worse than Base-MLP on full-text, and slightly better on T_{all}. Using all titles for training, every classifier is already able to outperform itself on EconBiz Full. On PubMed, however, the full-text performance is not reached.

C.5.2 MLP. In this section, we experiment with two changes to the MLP that are inspired by fastText [28]. First, we adopt the neural network architecture of fastText. Secondly, we add bi-grams as features.

Dataset		α	p	F_1-score	Dataset		α	p	F_1-score
EconBiz	Full	0.001	0.25	0.292	PubMed	Full	0.001	0.25	0.343
EconBiz	Full	0.001	0.50	0.327	PubMed	Full	0.001	0.50	0.408
EconBiz	Full	0.001	0.75	**0.332**	PubMed	Full	0.001	0.75	**0.423**
EconBiz	Full	0.010	0.25	0.192	PubMed	Full	0.010	0.25	0.242
EconBiz	Full	0.010	0.50	0.268	PubMed	Full	0.010	0.50	0.284
EconBiz	Full	0.010	0.75	0.285	PubMed	Full	0.010	0.75	0.316
EconBiz	Full	0.100	0.25	0.107	PubMed	Full	0.100	0.25	0.218
EconBiz	Full	0.100	0.50	0.107	PubMed	Full	0.100	0.50	0.218
EconBiz	Full	0.100	0.75	0.107	PubMed	Full	0.100	0.75	0.218
EconBiz	T1	0.001	0.25	0.283	PubMed	T1	0.001	0.25	0.329
EconBiz	T1	0.001	0.50	0.320	PubMed	T1	0.001	0.50	0.389
EconBiz	T1	0.001	0.75	**0.322**	PubMed	T1	0.001	0.75	**0.411**
EconBiz	T1	0.010	0.25	0.187	PubMed	T1	0.010	0.25	0.246
EconBiz	T1	0.010	0.50	0.256	PubMed	T1	0.010	0.50	0.282
EconBiz	T1	0.010	0.75	0.288	PubMed	T1	0.010	0.75	0.310
EconBiz	T1	0.100	0.25	0.112	PubMed	T1	0.100	0.25	0.218
EconBiz	T1	0.100	0.50	0.112	PubMed	T1	0.100	0.50	0.218
EconBiz	T1	0.100	0.75	0.107	PubMed	T1	0.100	0.75	0.218
EconBiz	T_{all}	0.001	0.25	0.290	PubMed	T_{all}	0.001	0.25	0.306
EconBiz	T_{all}	0.001	0.50	0.370	PubMed	T_{all}	0.001	0.50	0.377
EconBiz	T_{all}	0.001	0.75	**0.392**	PubMed	T_{all}	0.001	0.75	**0.424**
EconBiz	T_{all}	0.010	0.25	0.108	PubMed	T_{all}	0.010	0.25	0.240
EconBiz	T_{all}	0.010	0.50	0.175	PubMed	T_{all}	0.010	0.50	0.264
EconBiz	T_{all}	0.010	0.75	0.252	PubMed	T_{all}	0.010	0.75	0.293
EconBiz	T_{all}	0.100	0.25	0.112	PubMed	T_{all}	0.100	0.25	0.215
EconBiz	T_{all}	0.100	0.50	0.108	PubMed	T_{all}	0.100	0.50	0.216
EconBiz	T_{all}	0.100	0.75	0.110	PubMed	T_{all}	0.100	0.75	0.215

Table 11: Results of grid-search for CNN on EconBiz and PubMed. α denotes the learning rate, and p denotes the keep probability of dropout.

fastText architecture. We here analyze if adopting design decisions from fastText [28], which has gained a lot of popularity recently due to its open source code, can improve the results. The neural network architecture of fastText is similar to Base-MLP in that it is a fully-connected feed-forward neural network with one hidden layer. However, there are a few key differences. In fastText, the activation function on the hidden layer is the identity function, as opposed to the rectifier activation in Base-MLP. Therefore, fastText is a linear classifier, whereas Base-MLP is a non-linear classifier. This makes fastText less powerful in the sense that it is not able to identify interdependencies between the input features. However, the strength of fastText lies in the fact that it has strong performance while still being computationally efficient. To this end, fastText employs hierarchical softmax instead of the more costly softmax to compute the probabilities at the output layer. However, since we employ sigmoid at the output layer, which is computationally faster than softmax, this is not a concern in our study.

Another difference is that fastText does not employ dropout. We do not know if this is a deliberate choice or whether dropout was simply not needed because other regularization techniques worked sufficiently well. However, since another grid-search for the optimal dropout value is computationally too costly, we will stick with the fastText design, and not employ dropout initially. However, in order to make sure that any difference between the fastText design and our design does not result from insufficient regularization, we additionally conduct an experiment with keep probability $p = 0.5$, which was the best configuration in Base-MLP (see Section C.4).

To assess whether the design decision by fastText may improve the results of our model, we conduct experiments where we employ the identity function at the hidden layer and set the keep probability to $p = 1$. We experiment with several hidden layer sizes, including 10, 100, and 200, which were all used in the original fastText paper [28]. However, since the number of classes in our study is much larger, and due to the fact that multi-label classification is arguably a harder task than multi-class classification, we also experiment with larger hidden layer sizes up to 1,000 units.

The results in Table 14 show that we generally achieve better results with more hidden units. While increasing the number of hidden units from 10 to 200 yields a rapid increase in performance, the improvement when increasing from 200 units to 1,000 is not very large, indicating a possible convergence at 1,000 units. Comparing to the results of Base-MLP (last row in Table 14, we can observe that the best fastText configuration without dropout is between 0.012 and 0.036 points in F_1-score worse than Base-MLP. Using

Dataset		α	p	F_1-score	Dataset		α	p	F_1-score
EconBiz	Full	0.001	0.25	0.332	PubMed	Full	0.001	0.25	0.474
EconBiz	Full	0.001	0.50	0.328	PubMed	Full	0.001	0.50	0.509
EconBiz	Full	0.001	0.75	0.321	PubMed	Full	0.001	0.75	**0.514**
EconBiz	Full	0.010	0.25	**0.354**	PubMed	Full	0.010	0.25	0.399
EconBiz	Full	0.010	0.50	0.308	PubMed	Full	0.010	0.50	0.424
EconBiz	Full	0.010	0.75	**0.354**	PubMed	Full	0.010	0.75	0.468
EconBiz	Full	0.100	0.25	0.121	PubMed	Full	0.100	0.25	0.331
EconBiz	Full	0.100	0.50	-	PubMed	Full	0.100	0.50	0.390
EconBiz	Full	0.100	0.75	0.239	PubMed	Full	0.100	0.75	0.395
EconBiz	T1	0.001	0.25	0.315	PubMed	T1	0.001	0.25	0.399
EconBiz	T1	0.001	0.50	0.330	PubMed	T1	0.001	0.50	0.426
EconBiz	T1	0.001	0.75	0.317	PubMed	T1	0.001	0.75	0.432
EconBiz	T1	0.010	0.25	0.338	PubMed	T1	0.010	0.25	0.358
EconBiz	T1	0.010	0.50	**0.354**	PubMed	T1	0.010	0.50	0.419
EconBiz	T1	0.010	0.75	0.338	PubMed	T1	0.010	0.75	**0.434**
EconBiz	T1	0.100	0.25	0.176	PubMed	T1	0.100	0.25	0.264
EconBiz	T1	0.100	0.50	0.239	PubMed	T1	0.100	0.50	0.300
EconBiz	T1	0.100	0.75	0.233	PubMed	T1	0.100	0.75	0.320
EconBiz	T_{all}	0.001	0.25	0.413	PubMed	T_{all}	0.001	0.25	0.440
EconBiz	T_{all}	0.001	0.50	**0.438**	PubMed	T_{all}	0.001	0.50	0.476
EconBiz	T_{all}	0.001	0.75	0.435	PubMed	T_{all}	0.001	0.75	**0.490**
EconBiz	T_{all}	0.010	0.25	0.315	PubMed	T_{all}	0.010	0.25	0.331
EconBiz	T_{all}	0.010	0.50	0.377	PubMed	T_{all}	0.010	0.50	0.389
EconBiz	T_{all}	0.010	0.75	0.392	PubMed	T_{all}	0.010	0.75	0.391
EconBiz	T_{all}	0.100	0.25	0.161	PubMed	T_{all}	0.100	0.25	0.240
EconBiz	T_{all}	0.100	0.50	0.219	PubMed	T_{all}	0.100	0.50	0.275
EconBiz	T_{all}	0.100	0.75	0.243	PubMed	T_{all}	0.100	0.75	0.295

Table 12: Results of grid-search for LSTM on EconBiz and PubMed. α denotes the learning rate, and p denotes the keep probability of dropout.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
Base-MLP	**0.439**	**0.396**	**0.475**	**0.527**	**0.479**	0.486
Base-CNN	0.332	0.322	0.392	0.423	0.411	0.424
Base-LSTM	0.354	0.330	0.438	0.514	0.432	**0.490**

Table 13: Results on EconBiz and PubMed for base models.

$p = 0.5$, the performance goes up slightly in 4 out of 6 cases. Yet, the performance of Base-MLP can still not be reached. Hence, in subsequent experiments, we will keep Base-MLP unchanged.

Adding Bi-grams as Features. MLPs for text have the drawback that due to the fixed-size input they cannot retain word order out-of-the-box. In order to retain some sense of word order, it is a common approach to consider n-grams for the bag-of-words approach. For instance, fastText produces better results on all datasets considered in their study when adding bi-grams [28]. Hence, we test the effect of adding bi-grams in an experiment here.

In our experiments, we employ the Base-MLP architecture, but add the 25,000s most frequent bi-grams in addition to the unigrams.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
10	0.107	0.207	0.208	0.326	0.301	0.285
50	0.380	0.335	0.391	0.449	0.412	0.411
100	0.398	0.357	0.420	0.479	0.439	0.446
200	0.408	0.364	0.433	0.495	0.453	0.463
300	0.411	0.366	0.432	0.501	0.456	0.470
500	0.409	0.367	0.439	0.505	0.459	0.473
1,000	0.411	0.366	0.439	0.506	0.461	0.474
1,000 (p = 0.5)	0.427	0.383	0.438	0.509	0.466	0.467
Base-MLP	**0.439**	**0.396**	**0.475**	**0.527**	**0.479**	**0.486**

Table 14: Results on EconBiz and PubMed with MLP-Base but using identity as activation function on the hidden layer, and no dropout (p = 1). The rows indicate the number of hidden units used. For comparison, the last two rows show results with dropout (p = 0.5), and the results of Base-MLP from Table 13.

Please view Table 15 for the results of the experiments. Adding bi-grams helps in almost all cases, most notably on the large title

dataset, where they boost the performance by up to 0.018 in F_1-score. However, the small title datasets do not benefit from bi-grams, and the performance even suffers on EconBiz.

However, since the full-text and large title datasets benefit, in subsequent experiments, we will include bi-grams in all experiments with MLP.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
Unigrams	0.442	**0.396**	0.475	0.527	**0.479**	0.486
Bi-grams	**0.451**	0.373	**0.493**	**0.528**	**0.479**	**0.491**

Table 15: Results on EconBiz and PubMed with Base-MLP but adding the 25,000 most frequent bi-grams as features. The results for "Unigrams" are taken from the results in Table 13.

C.5.3 CNN. In this section, we experiment with two changes to the CNN that have emerged from a study on using a CNN for extreme multi-label classification by Liu et al. [51]. First, we experiment with dynamic max-pooling. Secondly, we assess the effect of adding a bottleneck layer to the network.

Dynamic Max-Pooling. We test the effect of dynamic max-pooling on the performance of the CNN. The relevant hyperparameter here is p, which determines the number of chunks to split the detector stage into. Unfortunately, in their paper, Liu et al. [51] do not explicitly state their choice of p or whether it was determined experimentally on a validation set. Therefore, we gradually increased p starting from $p = 2$ until we did not observe a considerable improvement in any of the experiments anymore, which was at $p = 3$.

Please see the results in Table 16. First, we can observe that dynamic max-pooling benefits the CNN on full-texts, but slightly hurts its performance on titles. Secondly, the magnitude of the effect of dynamic max-pooling depends on the dataset. While the differences in performance between different number of chunks are rather small on the EconBiz dataset, they are comparably larger on PubMed. In both cases, however, the full-text benefits from more chunks, especially on PubMed, where the scores go up by 0.026 in F_1-score with $p = 2$. When increasing p even further from $p = 2$ to $p = 3$, the additional gain is marginal (0.002 on EconBiz and 0.001 on PubMed). Hence, we stopped increasing p at that point.

In subsequent experiments, we will continue with $p = 3$ for full-texts and $p = 1$ for titles.

Bottleneck Layer. We evaluate the effect of injecting a bottleneck layer after the pooling stage. Besides reducing the number of parameters and hence making efficient training with GPUs possible on datasets with large label spaces, this has shown some beneficial generalization effect in the study by Liu et al. [51]. Following that study, we experiment with 500 units as the size of the bottleneck layer. In order to investigate if widening the bottleneck layer even further improves generalization, we also experiment with 1,000 units.

However, if the CNN has 100 filters and three different window sizes, the output size after the pooling stage is 300, so a layer with

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
p = 1	0.332	**0.322**	**0.392**	0.423	**0.411**	**0.424**
p = 2	0.338	0.315	0.387	0.449	0.408	0.415
p = 3	**0.340**	0.314	0.391	**0.450**	0.409	0.414

Table 16: Results on EconBiz and PubMed with CNN using different values for the number of chunks p. The values for $p = 1$ stem from Table 13.

500 units is not a bottleneck layer anymore since the number of nodes increases compared to the pooling stage. To address this issue, we also experiment with 250 units on the bottleneck layer. To see that a bottleneck layer with 250 units actually decreases the number of parameters, consider the following exemplifying calculation with EconBiz, where the number of labels is approximately 6,000. Without a bottleneck layer, the number of weights between the pooling stage and the output layer is approximately (that is, neglecting the bias) $300 \cdot 6{,}000 = 1{,}800{,}000$. If a bottleneck layer with 250 units is added, the number of weights is approximately $300 \cdot 250 + 250 \cdot 6000 = 1{,}575{,}000$.

The results can be found in Table 17. Injecting a bottleneck layer has a considerable uniformly beneficial effect on the EconBiz dataset. The performance increases by between 0.011 and 0.031 in F_1-score when a bottleneck layer with 500 units is added. The gain is smaller with 250 units. Transitioning from 500 to 1,000 units, the difference has a minor effect by 0.003 to 0.004 points in F_1-score on the full-text. On the title datasets, the difference is barely noticeable. This observation also holds on the PubMed dataset. However, in contrast to the EconBiz dataset, the large title dataset does not benefit from adding a bottleneck layer. Here, a bottleneck layer of 500 or 1,000 units has a marginal negative effect. The full-text and small title datasets are pushed by an F_1-score of 0.025 and 0.011, respectively.

Going forward, we will use a bottleneck layer of 1,000 units in all full-text experiments because it clearly yields the best performance on both datasets. For the title experiments, we choose a bottleneck layer of 500 units uniformly for simplicity. We argue that this is justified even though 500 units does not yield the best performance on the large PubMed title dataset since the difference is very marginal (0.001).

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
none	0.340	0.322	0.392	0.450	0.411	**0.424**
250	0.362	0.336	0.396	0.462	0.414	0.416
500	0.371	**0.341**	**0.403**	0.475	**0.422**	0.423
1,000	**0.374**	0.340	**0.403**	**0.477**	**0.422**	0.422

Table 17: Results on EconBiz and PubMed with CNN using a bottleneck layer. The results from the experiment without bottleneck layer usage are carried forward from Table 16.

C.5.4 LSTM. In this section, we present experimental results for three changes to our base LSTM. First, we evaluate different methods for aggregating the outputs of the LSTM. Secondly, we test the effect of OverEager LSTMs. Finally, we employ bidirectional LSTMs.

Aggregation Method. We compare four different strategies how to aggregate the outputs of the LSTM. Besides taking the average as done in the base model, we conduct experiments with the strategies to sum all outputs, to pick the last output, or to use attention to obtain a weighted sum of the outputs. The attention mechanism was formally introduced in Section B.2.3.

The results are displayed in Table 18. Attention is the winner in 4 out of 6 experiments and places second in the remaining two. However, the lead over the second best method is rather small in 3 out of those 4 experiments where attention wins, not exceeding a gain of 0.003 in F_1-score. On EconBiz Full, however, the gain is as much as 0.022. On the remaining two datasets, the summation of outputs performs best. Again, on one of these two the lead over the second best method (attention) is as small as 0.001 in F_1-score. On the other hand, on PubMed T1, a considerable gain of 0.006 can be observed.

With the exception of attention, which yields consistently good results on all datasets, we can see that the results are not very consistent across different datasets. For example, the "last" method performs very poorly on EconBiz Full, but outperforms the third and fourth best methods on PubMed T_{all} by a considerable margin.

In subsequent experiments we will use attention on all datasets since it places best in 4 out of 6 (sub-)datasets and has a very marginal difference to the best method on EconBiz T_{all}. Due to its robust results, we argue that it is the best candidate to use as aggregation method going forward.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
average	0.354	0.354	0.438	0.514	0.434	0.490
sum	0.330	0.359	**0.440**	0.479	**0.452**	0.489
last	0.207	0.346	0.437	0.497	0.445	0.498
attention	**0.377**	**0.360**	0.439	**0.516**	0.446	**0.501**

Table 18: Results on EconBiz and PubMed with LSTM using different aggregation methods. The values for 'average' stem from Table 13.

OverEager LSTMs. We test OE-LSTMs by employing them with each aggregation method. The results are shown in Table 19. For the sake of clarity, we included the results from Table 18. On the title datasets, OE-LSTMs yield a considerable benefit. On EconBiz T_{all}, the best aggregation method (sum) for the non-OverEager LSTM performs 0.023 points in F_1-score worse than the same aggregation method with OE-LSTM. On PubMed T_{all}, that difference is 0.008 points (attention). On full-texts, there is no considerable benefit. Instead, on EconBiz, the performance even decreases by 0.009 points. It is noteworthy that OE-LSTMs seem to struggle more to train properly than non-OE-LSTMs do. This can be seen by the fact that three experiments yield results below the very poor 0.300 mark, as opposed to only one experiment in non-OE-LSTMs.

Although the results are very promising for titles, we will not employ OverEager LSTMs in subsequent experiments. The reason is that, although the method can technically be applied to full-texts as well, in practice it is only effective on titles. This is due to the length of full-text documents, which exceed the maximum length of 250 in almost all cases. In order to include this method in a future study, it requires some modification, which we discuss in Section E.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
average	0.354	0.354	0.438	0.514	0.434	0.490
sum	0.330	0.359	0.440	0.479	0.452	0.489
last	0.207	0.346	0.437	0.497	0.445	0.498
attention	**0.377**	0.360	0.439	0.516	0.446	0.501
OE-average	0.368	0.364	0.458	**0.517**	0.465	0.507
OE-sum	0.106	**0.392**	**0.463**	0.493	**0.472**	0.506
OE-last	0.275	0.108	0.443	0.504	0.447	0.497
OE-attention	0.358	0.368	0.458	0.509	0.466	**0.508**

Table 19: Results on EconBiz and PubMed with LSTM using OE-LSTMS with several aggregation methods. The values for 'average', 'sum', 'last', and 'attention' stem from Table 18.

Effect of Bidirectional LSTMs. We study the effect of bidirectionality by comparing against unidirectional LSTMs, as shown in Table 20. Bidirectional LSTMs improve the results considerably on both datasets. On EconBiz, both the full-text performance and the title performance (on T_{all}) improves. On PubMed, the full-text seems not affected at all by bidirectionality, but the titles benefit mildly.

Consequently, we will employ bidirectional LSTMs in the subsequent experiments.

Dataset	EconBiz			PubMed		
Experiment	Full	T1	T_{all}	Full	T1	T_{all}
unidirectional	0.377	**0.360**	0.439	**0.516**	0.446	0.501
bidirectional	**0.386**	0.357	**0.452**	**0.516**	**0.460**	**0.505**

Table 20: Results on EconBiz and PubMed with LSTM using a bidirectional LSTM. The values for 'unidirectional' stem from the row labeled 'attention' in Table 18.

C.6 Effect of Capacity

Because in this study we deal with very large title datasets, one approach to push title performance is to increase the representational capacity of the model.

However, increasing the effective capacity is not in a one-to-one relation with increasing the representational capacity of the model in terms of number of learnable parameters (for an introduction to this terminology, please refer to Goodfellow et al. [27, p. 113ff.]). Oftentimes, adding an additional layer to the network is more effective than increasing the width of the existing layers, even though increasing the width effectively adds more parameters to

the model. For example, in Section C.5.3, we have even seen that a bottleneck layer that reduces the total number of parameters improves the generalization performance of the model. Although an MLP with a single hidden layer can represent any function if provided with enough units [36], in recent years a lot of success has been achieved with increasing the depth of convolutional neural networks, especially in the vision domain [32]. These empirical results have theoretical foundation for MLPs, which suggest that depth is preferable over width when the number of parameters is fixed [21, 85].

Therefore, in this section, we will follow both the approaches of making the models wider and making the models deeper.

C.6.1 MLP. We explore several ways to increase the capacity of the MLP. As discussed before, increasing both depth or width of the MLP may help. In our experiments, we gradually increase the number of neurons by 500 in each step until we do not observe any considerable gains in performance.

Similarly, we gradually increase the number of layers (with 1,000 units each) until we no longer observe performance gains. Since it is known that deeper neural networks are harder to optimize due to the exploding and vanishing gradient problems, we experiment with SNNs and Batch Normalization as two strategies to alleviate this problem. For SNNs, we test the values $p \in \{0.95, 0.90\}$ for the keep probability of alpha-dropout, as suggested by the inventors of SNNs [44] (normal dropout is no longer applied in SNNs). Regarding Batch Normalization, it has been reported that the generalization capabilities of Batch Normalization are sometimes strong enough to replace dropout [39]. Therefore, we experiment with keep probabilities in $\{1, 0.5\}$.

Please consider the results of the experiments displayed in Table 21. Generally, we may observe that widening the hidden layer is beneficial in all cases except on EconBiz T1, where the performance drops by 0.01 in F_1-score when increasing from 1,000 to 1,500 hidden nodes, and by another 0.007 when going up to 2,000 units. On all other datasets, widening the layer does not harm the performance, yielding considerable gains of 0.009 and 0.006 on EconBiz T_{all} and EconBiz Full, respectively. On PubMed, the gain is smaller, but PubMed T_{all} still benefits with an increase of 0.004.

Deeper architectures do much worse than the wide architectures in our experiments. With the one exception being PubMed T_{all}, the performance drops with increasingly deeper MLPs. This holds for the deep models with plain rectifier activation function, as well as for the models with integrated SELU or Batch Normalization. While the different p values for alpha-dropout generally do not have a large effect on the outcome of our experiments, the differences for the two Batch Normalization configurations are quite large. A dropout rate of 0.5 is vital to the success of the MLP on the smaller datasets in EconBiz, where an increase by between 0.039 and 0.072 must be attributed to its presence. On the other datasets, the presence of dropout has a smaller positive effect (approximately 0.008 on EconBiz T_{all}), or even a large negative impact of up to 0.05 in F_1-score, as seen on PubMed T_{all}. On the latter dataset, the two-layer network with Batch Normalization even performs best out of all methods, outperforming the best wide architecture by 0.009.

Subsequently, we will use the widest architecture for all datasets but PubMed T_{all}, where we will use a two-layer MLP with Batch Normalization and no dropout. Both configurations are chosen because they achieved the best outcome on the respective datasets.

	EconBiz			PubMed		
	F	T1	T_{all}	F	T1	T_{all}
(1,000)	0.451	**0.373**	0.493	0.528	**0.479**	0.491
(1,500)	0.455	0.363	0.498	0.529	**0.479**	0.493
(2,000)	**0.457**	0.356	**0.502**	**0.530**	**0.479**	0.495
2	0.433	0.367	0.455	0.487	0.445	0.433
3	0.407	0.356	0.401	0.448	0.400	0.383
2 + SNN (p = 0.9)	0.368	0.306	0.449	0.499	0.454	0.487
2 + SNN (p = 0.95)	0.372	0.309	0.451	0.500	0.454	0.485
3 + SNN (p = 0.9)	0.341	0.291	0.419	0.450	0.446	0.475
3 + SNN (p = 0.95)	0.347	0.285	0.440	0.446	0.446	0.475
2 + BN (p = 0.5)	0.430	0.372	0.476	0.492	0.450	0.466
2 + BN (p = 1)	0.377	0.305	0.468	0.486	0.449	**0.504**
3 + BN (p = 0.5)	0.408	0.356	0.454	0.472	0.423	0.445
3 + BN (p = 1)	0.336	0.317	0.451	0.467	0.419	0.496

Table 21: Results on EconBiz and PubMed with MLP using different strategies for increasing the capacity. The first three rows use wide layers, whereas the remaining 10 rows use multiple layers with 1,000 units and different normalization strategies. The values for (1,000) stem from Table 15.

C.6.2 CNN. In this section, we experiment with ways to increase the representational capacity of the CNN. As with MLPs, it would be an option to add additional convolutional layers and pooling layers to CNN, as is common in the vision domain. However, a previous study has found that introducing depth does not yield a benefit for text classification [46]. Instead, we follow two approaches to increase the width of the existing layer. First, we increase the set of window sizes to extract features from. Secondly, we widen the feature map. A study testing the sensitivity of the hyperparameters of CNNs by Zhang and Wallace [102] also suggests to carefully tune these parameters.

Window Sizes. The literature does not provide an explanation how to choose window sizes. In fact, the window sizes employed for text classification vary strongly. While Kim [42] chooses 3,4, and 5, Liu et al [51] choose 2, 4, and 8 for their study on extreme multi-label classification. In the following experiments, we test both these values. However, the intuition behind employing different window sizes in the first place is that these also capture different features. Therefore, it is reasonable to assume that the combination (2,3,4,5,8) of window size will be able to extract even more diverse features, and thus yield even better results.

Please see results in Table 22. The solution (2,4,8) by Liu et al. [51] and the solution (3,4,5) by Kim [42] perform comparably well. While (3,4,5) has a slight advantage on EconBiz Full and PubMed T_{all}, (2,4,8) performs better on the other sub-datasets. The combination (2,3,4,5,8) of the two solutions, however, is clearly the best on all datasets. Hence, we will use this combined solution in subsequent experiments.

	EconBiz			PubMed		
	F	T1	T_{all}	F	T1	T_{all}
(3,4,5)	0.374	0.341	0.403	0.477	0.422	0.424
(2,4,8)	0.369	0.347	0.408	0.478	0.424	0.422
(2,3,4,5,8)	**0.383**	**0.356**	**0.414**	**0.485**	**0.430**	**0.431**

Table 22: Results on EconBiz and PubMed with CNN using different values for the window sizes. The values for (3,4,5) are copied from Table 17.

Larger Feature Map Size. As explained in Section B.2.2, each filter can be thought of as a feature extractor that is swiped over the text. Naturally, the more filters the CNN is provided (or synonymously: the larger the feature map size), the more features it is able to extract. Hence, we gradually increase the number of filters until no additional improvement is observed. In the past, using around 100 filters has worked well in many cases[42, 51]. Therefore, we explore the space around 100 more fine-grained, and increase the filter size more drastically for larger filter sizes. We argue that this is a reasonable trade-off between computational complexity and adequate examination of the hyperparameter space. In particular, we have evaluated the feature map sizes 100, 125, 150, 200, 400, and 800.

The results are shown in Table 23. Using 400 filters yields the best performance on 5 out of 6 datasets. When the number of filters is increased even further, the performance drops on all datasets. Up to 400 filters, we can observe a steady improvement in performance in almost all cases. The only exception is PubMed Full, which shows its best performance at filter size 100. Hence, in subsequent experiments, we will use 400 filters on all datasets but PubMed Full, where we will use 100 filters.

	EconBiz			PubMed		
	F	T1	T_{all}	F	T1	T_{all}
100	0.383	0.356	0.414	**0.485**	0.430	0.431
125	0.380	0.357	0.419	0.484	0.431	0.432
150	0.385	0.357	0.417	0.483	0.433	0.434
200	0.385	0.362	0.421	0.465	0.435	0.437
400	**0.388**	**0.365**	**0.426**	0.439	**0.439**	**0.440**
800	0.384	0.359	0.426	0.429	0.431	0.430

Table 23: Results on EconBiz and PubMed with CNN using different feature map sizes. The values for 100 are copied from Table 22.

C.6.3 LSTM. In many NLP tasks, it has shown to be beneficial to stack multiple layers of LSTMs. However, training can sometimes be harder, and overfitting occurs more easily. Dropout is a common way to reduce the risk of overfitting. However, as described in Section B.2.3, using the default dropout technique on the recurrent connections is discouraged because it distorts the recurrent net's ability to learn connections between time steps. Instead, Gal et al. [23] propose a different kind of dropout called variational dropout, which in their experiments helps for deep networks in particular.

Similar to how we proceed with MLPs, we explore two directions how to increase the capacity of the LSTM. First, we present the effect of increasing the size of the memory cell, which corresponds to increasing the width of the network. Since using 512 units in the memory cell already yields good results, we increased the number of hidden units moderately at first to avoid advancing too quickly into cell sizes that can easily overfit the data. We then increased the size more quickly as we progressed. In the end, we evaluate the unit sizes 768, 1,024, 1,536, and 2,048. We stopped at 2,048, because the increase in performance over 1,536 was not very large, but the training time was becoming unacceptable.

Secondly, we keep the number of units in the memory cell at 512, but add one more layer. Since we observed that this does not improve the performance over the configuration with one LSTM layer, we performed additional experiments where we use variational dropout to prevent overfitting.

The results of these experiments are displayed in Table 24. For the larger part, the performance of the LSTM benefits from an increase in memory cell size. On all datasets except EconBiz Full and EconBiz T1, the F_1-score increases steadily with the memory cell size. When increasing from 1,536 to 2,048 units, the improvement is relatively small, especially on the large title datasets (an increase by not more than 0.002). On EconBiz T1, the performance even becomes worse. On EconBiz Full, the LSTM generally fails to learn for larger memory cells, performing very poorly or even running into numerical errors during training (displayed as F_1 score "0"). For memory cells with size larger than 1,536, we did not conduct experiments because the model did not fit into the GPU memory.

The LSTM does not benefit from an additional layer in the same way as from a larger memory cell. Here, only PubMed T_{all} benefits from the additional layer. However, the benefit is only 0.003 and thus much smaller than the overall benefit from increasing the width, which is 0.011 in F_1-score. Adding variational dropout does not help in any case. In fact, in 4 out of 6 cases, the performance either drops drastically or the LSTM fails to learn in the first place, resulting in numerical errors. In the remaining two cases, that is, on both PubMed title datasets, the performance degrades mildly in comparison, by up to 0.017.

As stated before, the time for training a model increases drastically when increasing the memory cell size. For example, while training the model on PubMed T_{all} takes about a day for a cell size of 1,536, it takes several days when increasing the size to 2,048. This can be explained by the fact that the total number of learnable parameters in the model is a quadratic function in the memory cell size m. In contrast, the improvement in performance is marginal on both T_{all} sub-datasets. Considering that we do a 10-fold cross-validation in our final experiments, we decided to use a memory cell of 1,536 units for our final experiments on title datasets to evaluate our models in reasonable time. On PubMed Full and EconBiz Full, we will set $m = 1024$ and $m = 512$, respectively, as these values yield the best results for the respective datasets.

	EconBiz			PubMed		
	Full	T1	T_{all}	F	T1	T_{all}
512	**0.386**	0.357	0.452	0.516	0.460	0.505
768	0	0.358	0.456	0.523	0.462	0.509
1024	0.121	0.356	0.464	**0.525**	0.462	0.512
1536	-	**0.369**	0.469	-	0.464	0.515
2048	-	0.363	**0.471**	-	**0.467**	**0.516**
(512,512)	0.292	0.346	0.448	0.494	0.453	0.508
(512,512) + VD	0	0.148	0.140	0.223	0.450	0.491

Table 24: Results on EconBiz and PubMed with LSTM using different ways to increase capacity. In the first 5 rows, the memory cell size is altered. The values for 512 stem from Table 20. In the remaining two rows, an additional layer is added. "VD" refers to the use of variational dropout.

C.7 Experiments with Neural MetaLabeler

We measure the effect of different configurations of Neural MetaLabeler in our experiments. Since the method in its neural implementation is a novel technique developed in this study, there is no previous literature that could hint at how to choose the hyperparameter γ. Unfortunately, again due to the large datasets assessed in this work, it is impractical to do an exhaustive search for the best γ value. Hence, we argue that it makes intuitively sense to evaluate those scenarios where predicting the correct number of labels has low priority, equal priority, and high priority relative to optimizing the multi-label objective. Conducting some preliminary experiments on EconBiz T1 with the MLP, we found that $\gamma \in \{0.1, 0.5, 0.9\}$ are reasonable choices. This can be seen by inspecting the loss associated with predicting the number of labels after training has concluded. In these preliminary experiments, they finish at approximately 1.22, 0.26, and 0.05 for $\gamma = 0.1, 0.5, 0.9$, respectively. These losses show that the γ values we chose yield drastically different emphases that the neural network puts on predicting the number of labels correctly.

In the experiments, whose results can be inspected in Table 25, we used only the MLP. We evaluated each combinations of the two NML variants ϕ_c and ϕ_s, and the above values for γ.

Considering the results, the NML methods do not have a large impact with respect to the research question of our study. On 2 out of 6 datasets, the threshold method yields the best results. On EconBiz T1 and PubMed T1, the improvement of the best NML method over the threshold method is 0.03, which is relatively large. However, when looking at the performance on the respective T_{all} datasets, this advantage vanishes, and the threshold method outperforms NML slightly by 0.001 and 0.002 in F_1-score, respectively. On the full-text datasets, again one of the NML methods performs best. However, in contrast to the T1 experiments, the gain is relatively small (0.001 on EconBiz and 0.003 on PubMed).

When comparing the NML configurations with each other, we can observe that none of the configurations outperforms the others consistently. Except for $\phi_s(0.9)$, which is consistently the worst configuration on EconBiz and never the best performer on PubMed, the scores of the different configurations are often fairly close. Moreover, each of the configurations except $\phi_s(0.9)$ is the top performer on at least one sub-dataset.

We decided against employing NML for CNNs and LSTMs as well for several reasons. First, the role of the hyperparameter γ in NML is not well understood yet. In Section D, we provide more detail on this issue. As argued before, in order to achieve a fair comparison between titles and full-text, a good understanding of how to set the hyperparameter on each dataset is required. Secondly, conducting an experiment on all 6 subsets and all 6 configurations results in 36 experiments for each of the neural networks CNN and LSTM. Since in particular the LSTM takes a long time to train (more than a day), this falls out of scope for this study. Hence, in order to have a fair comparison also between the types of neural networks, we will not use NML for our final experiments.

		EconBiz			PubMed		
		Full	T1	T_{all}	Full	T1	T_{all}
MLP	θ	0.457	0.356	**0.502**	0.530	0.449	**0.504**
	$\phi_c(0.1)$	**0.458**	0.372	0.492	0.531	0.476	0.502
	$\phi_c(0.5)$	**0.458**	0.373	0.501	0.531	0.475	0.502
	$\phi_c(0.9)$	0.457	**0.386**	0.497	0.525	0.466	0.500
	$\phi_s(0.1)$	0.457	0.372	0.488	**0.533**	**0.479**	0.501
	$\phi_s(0.5)$	0.455	0.371	0.490	0.532	**0.479**	0.502
	$\phi_s(0.9)$	0.382	0.325	0.462	0.523	0.473	0.498

Table 25: Comparison of different MetaLabeler configurations for EconBiz and PubMed with MLP using different configurations of NML. The values for θ stem from Table 21.

D DISCUSSION

In this section, we review and discuss some of the results from the experiments we conducted in Section C.

Non-linearity is critical for the MLP. Our experiments with the fastText architecture showed considerably worse results than the base model. The observed results are not unexpected. As stated before, Base-MLP is a non-linear classifier in contrast to when "identity" is employed at the hidden layer. This enables it to take into account simultaneous occurrence of words for making the classification decision. The fact that this boosts the performance considerably is some evidence that relationships between words are to some extent important for text classification. The study by Galke et al. [24] has shown a similarly large difference between a non-linear MLP and a linear statistical classifier like logistic regression. It is not clear how much of the speed-up presented in the fastText paper [28] results from omitting a non-linear activation function. Future work could study the effect of employing a non-linear activation function on both classification performance and speed, and discuss their trade-off. However, this is beyond the scope of this paper.

The effect of bi-grams may depend on the number of words. The experiments with bi-grams have shown that including bi-grams helps on all datasets but EconBiz T1. It is not immediately clear why adding bi-grams does well on the full-text and on large title

datasets but does poorly on the small title datasets. However, an explanation could be that for Full and T_{all} the total number of words available for training is much larger than for T_1. While the number of combinations thus increases for Full and T_{all} compared to T_1, so does the absolute number of occurrences per bi-gram. Hence, during training the algorithm is presented with more cases per bi-gram where the bi-gram count is non-zero. While this effect also applies to unigrams, it is more prominent in bi-grams because their frequency is generally lower.

Dynamic max-pooling is reasonable only on full-text. Our experiments have clearly shown that dynamic max-pooling improves the performance when employed on the full-text, but rather hurts the performance when employed on titles. This can be explained by the length of the respective type of text. For titles, extracting only one maximum value from the text in the pooling stage is sufficient because the text is short and is therefore unlikely to contain multiple positions that indicate the same feature. By increasing the number of chunks, however, we add complexity to the model and thus decrease statistical efficiency for each parameter.

Full-texts, on the other hand, are more likely to contain multiple positions that indicate the same feature. Here, it is intuitive to split up the text into multiple chunks and handle these differently.

The bottleneck layer behaves inconsistently. In our experiments, the bottleneck layer has improved the performance on all datasets except PubMed T_{all}. Regarding the full-text datasets, these results are in some sense consistent with the previous experiments where we added dynamic max-pooling, and only the full-text benefited greatly from larger representational capacity. However, while there was an obvious explanation why dynamic max-pooling does not work for titles, it is not clear why the PubMed title datasets do not benefit from a bottleneck layer in the same way as full-texts do. This is particularly unexpected because PubMed T_{all} has the largest number of training samples and should thus be able to benefit from larger capacity the most.

On the other hand, on almost all the sub-datasets except PubMed T_{all}, adding a bottleneck layer with only 250 units improves the overall performance, even though this reduces the total number of parameters. Hence, these results support the hypothesis by Liu et al. [51] that an additional layer after the pooling stage adds hierarchical representational power.

In summary, these findings are rather contradictory, and in order to give an explanation of the bottleneck layer's behavior, more experiments are required.

Training LSTMs is difficult. In the experiments with LSTMs, we repeatedly observe that the model either yields very poor results or runs into numerical errors.

Consider the experiments that compare the aggregation methods (Table 18). Here, the "last" aggregation method seems to get stuck during training on EconBiz Full. Furthermore, the inconsistency in relative placings of the aggregation methods may suggest that random factors in the optimization process play a role in determining these placings. Attention, however, could be an exception since it consistently performs well. For a future study, it would be interesting to investigate if other variants of attention are as robust at the text classification task as the variant by Yang et al. [93] used in this study.

In the experiments regarding OE-LSTMs (Table 19), these training difficulties appear even more often. On EconBiz Full, increasing the number of layers or the memory cell size m both lead to very bad performances. What many of the cases where the LSTM fails to learn properly have in common is that they occur on EconBiz Full. Here, we set the learning rate to 0.01, in contrast to the other sub-datasets. While this was a reasonable choice considering the results from the grid-search as discussed in Section C.4.2, the learning rate may be too high for this dataset, so that small changes in the architecture break the optimization procedure.

However, the statement that LSTMs are difficult to train is also supported when considering the results where variational dropout is employed. In summary, our experience confirms previous literature on the topic [65].

Do convolutional networks need to be wider for text classification? In the title of their study, Le et al. [46] ask the question "Do Convolutional Networks have to be Deep for Text Classification?". They have shown that "shallow-and-wide" CNNs are superior to deep CNNs, and thus answer this question with "no". However, what they call "wide" is a CNN that uses three different window sizes and a feature map size of 100. In the study by Liu et al. [51], the feature maps are only slightly larger (up to 128). In contrast, our experiments have shown that increasing the width of the CNNs pushes the results. Combining the effect of taking into account 5 window sizes instead of 3, and the effect of increasing the feature map size from 100 to 400, the performance on EconBiz Full improves by 3.7%. Similarly, the performance on EconBiz T_{all} improves by 5.7%, and the score on PubMed T_{all} increases by 3.8%. On PubMed Full, we observe a gain of 1.7% by taking into account 5 window sizes. A sensitivity analysis of CNNs by Zhang and Wallace also recommends to explore the width more carefully [102]. Hence, instead of focusing on the depth, future research regarding CNNs for text classification could ask the question "Do Convolutional Networks have to be Wider for Text Classification?".

Wider networks do better than deeper networks. A general observation from our attempts to increase the capacity of the neural networks is that wider networks tend to work better than deeper networks in our study.

Regarding the MLP, in our final experiments, we employ a wide configuration on both types of text on EconBiz, and on PubMed Full as well. Even though we employed two strategies to facilitate learning in deep MLPs, Self-Normalizing Neural Networks and Batch Normalization, only one deep configuration made it into our final experiments. On PubMed T_{all}, the largest sub-dataset considered in our study, Batch Normalization seems to facilitate training successfully. Moreover, it is able to replace dropout as a regularizer. In order to verify this, we ran an additional experiment, where we use the same configuration but without Batch Normalization. Here, the score is 0.482 (as opposed to 0.504 with Batch Normalization). Hence, we suspect Batch Normalization plays a crucial role in obtaining the results with the two-layer MLP that outperform the wide architectures. However, it remains unclear why it does not have the same effect on the other datasets. A relevant factor could be the enormous size of the dataset.

Deep recurrent neural networks have so far only been successful for structured prediction tasks or sequence prediction tasks (see, for example, [71]), but to the best of our knowledge, there is no study showing that deep recurrent neural networks outperform shallow ones for text classification[17]. The rather bad results from our experiments where we stack two LSTM layers seem to confirm this. However, the even poorer performance of variational dropout may be explained by the possibility that the optimal learning rate is sensitive to whether variational dropout is applied or not. Unfortunately, as mentioned repeatedly earlier, optimizing the learning rate for each individual dataset is too costly for this work, especially because the datasets are very large and training even a single model can take more than one day. With this in mind, our results are by no means representative of the performance of deep LSTMs for text classification in general.

Neural MetaLabeler. The Neural MetaLabeler has shown some very good results on the T1 sub-datasets. For EconBiz T1 in particular, we determined reasonable values for the hyperparameter γ in beforehand, by verifying that the different values indeed result in different weights that the neural network puts on predicting the number of labels. However, on the larger T_{all} datasets, the NML was not able to outperform the threshold method. A possible explanation for this inconsistent behavior of NML could be given when taking another look at the loss associated with predicting the number of labels after training. On PubMed T_{all}, the losses after training with the three different gamma values $\gamma \in \{0.1, 0.5, 0.9\}$ are 2.72, 2.71, and 2.61, respectively. These values do not differ much from each other, and their absolute values are relatively large compared to 1.22, 0.26, and 0.05, which we found on EconBiz T1. This might indicate that the neural network does not put enough priority on predicting the correct number of labels to improve the results. On PubMed Full, the loss when using ϕ_s decreases from 2.10 to 2.04 to 0.80. In absolute terms, these are slightly lower numbers, indicating that the neural network puts more emphasis on predicting the number of labels, and in fact the performance improves slightly compared to the threshold method.

This analysis shows that NML may require a dataset specific careful tuning of the hyperparameter γ. Whether or not a well-tuned NML method can consistently outperform the threshold method is therefore an open question. Unfortunately, for this work an exhaustive examination of the hyperparameter space to answer this question is out of scope. Hence, we leave this question for future work.

Comparison of base models and final models. During development of our models, we aimed at improving both the performance on full-text and on the large title sub-datasets. In Table 26, we contrast the performance of the base models (see Table 13) we adopted from the literature with the models we used in the final experiments (gathered from the respective tables in Section C).

Overall, the changes we made to the base models were very beneficial. The final models improve upon the base models on both datasets and on both titles and full-texts. Only on T1 of PubMed and EconBiz, the MLP does worse than the baseline. This is comprehensible considering that T1 and T_{all} are trained with the same models that had been optimized towards performance on T_{all} during development. On all other sub-datasets, the performance improves, and often with large rates. For instance, the scores with MLP on the full-texts of EconBiz grew by 4.1% over the previous best performer from the study by Galke et al. [24], which we also used as the baseline for our study. For CNNs, the improvement is particularly large on the full-texts, where the F_1-score increased by 0.055 on EconBiz, and by 0.062 on PubMed.

Please consider the top performer among the base models on each dataset. On EconBiz, this is Base-MLP on both titles and full-texts. Here, the MLP, which is the top performer on EconBiz among the final models, increased the score of the titles by a larger absolute value than the score of the full-text, amplifying the advantage of training on the large title dataset. Analogously, the best performer on PubMed titles (LSTM, +0.025) has improved by more than the best performer on the full-texts of PubMed (MLP, +0.003). With respect to the research question in our study, these results support our presumption that complex deep learning models benefit particularly well from the large number of labeled titles. On the other hand, this does not always apply to all models. For example, the full-text models benefit more than the title models on both datasets when considering the CNN.

	EconBiz			PubMed		
	Full	T1	T_{all}	Full	T1	T_{all}
Base-MLP	0.439	0.396	0.475	0.527	0.479	0.486
MLP	0.457	0.356	0.502	0.530	0.449	0.504
± F_1-score	+0.018	-0.04	+0.027	+0.003	-0.03	+0.018
±%	+4.1	-10.1	+5.7	+0.6	-6.3	+3.7
Base-CNN	0.332	0.322	0.392	0.423	0.411	0.424
CNN	0.388	0.365	0.426	0.485	0.439	0.440
± F_1-score	+0.055	+0.043	+0.034	+0.062	-0.028	+0.016
±%	+16.9	+13.4	+8.7	+14.7	+6.8	+3.8
Base-LSTM	0.354	0.330	0.438	0.514	0.432	0.490
LSTM	0.386	0.369	0.469	0.525	0.464	0.515
± F_1-score	+0.032	+0.039	+0.031	+0.011	+0.032	+0.025
±%	+9.0	+11.8	+7.1	+2.1	+7.4	+5.1

Table 26: Results on EconBiz and PubMed for base models (see Table 13) in contrast to the scores of the models we used in the final experiments. The rows marked with "±F_1-score" show the absolute change of performance in terms of the F_1-score. The rows marked with "±%" show the relative change of the performance. Note that all scores result from evaluating on only one fold.

E FUTURE WORK

In this section, we discuss promising future research directions to pursue considering the results of this thesis. There are manifold research paths to follow. The first is concerned with the question whether the results from this study generalize to other contexts, i.e., datasets from other domains. A second direction could investigate some of the methods that were developed in this study, but whose effect could not be fully assessed. In a third direction, one could

[17] For an overview of deep LSTMs, one might refer to a blog post by Sebastian Ruder at http://ruder.io/deep-learning-nlp-best-practices/, accessed on February 5, 2018.

explore more techniques that benefit from particularly large sample sizes, and thus could potentially push the titles' performance in automatic semantic subject indexing even further.

E.1 Generalizability

In our study, we have shown that by making use of the surplus of labeled title data it is possible to almost reach or even outperform the full-text. To this end, we have conducted experiments on datasets from two scientific digital libraries, EconBiz (economics and business studies) and PubMed (biomedicine). In Section 6, we hypothesize that our results are likely to generalize to digital libraries from other scientific domains as well. This should be verified empirically by employing our developed deep learning classifiers to other scientific domains such as political sciences, or computer science.

Furthermore, in the discussion (Section 6) we argue that the full-texts of PubMed outperform the titles because the number of full-text just scratches on what previous studies found to be enough to let deep learning methods perform better than traditional models (650k). Since the number of publications increases every year, and the portion of open access publications rises as well, in a couple of years there will likely be full-text datasets with more than a million samples. It will be interesting to investigate if deep learning models benefit from this growth enough that title methods become clearly worse than full-texts again.

Finally, the question we ask in this study can be generalized to not only other scientific domains, but also to non-scientific text such as news articles or books. Abstracting the question even further, we can ask ourselves: Do relatively large datasets, where each sample has a low information density with respect to the prediction task, outperform relatively small datasets, where each samples has high information density? For instance, one could compare low-resolution images to high-resolution images, or short song snippets to the full length of the songs.

E.2 Analysis of Novel Methods From This Study

In this study, we developed two novel methods, Neural MetaLabeler and OverEager Long Short-Term Memory Networks. A close examination of these methods was out of the scope of this study, however, in this section, we outline what we would like to do with these techniques in future studies.

Neural MetaLabeler. Neural MetaLabeler is an adaptation of the multi-labeling technique MetaLabeler [84] for neural networks. Analogous to MetaLabeler, we proposed two variants that both aim to predict the number of labels a sample is likely to be assigned: The first operates on the prediction scores of the labels, and the other operates on the encoded features of the input text. The technique introduces a new hyperparameter that controls how much priority the prediction of the number of labels is given. Because this hyperparameter appears difficult to tune, we conducted experiments only with the MLP (see Section C.7). Here, the technique was very successful on the small title datasets, mildly successful on the full-texts of both datasets, and unsuccessful on the large title datasets. We observed that the choice of the hyperparameter on the large title datasets and on the full-texts was likely suboptimal, so the technique did not put enough priority on learning to predict the number of labels.

In a future study, we would like to conduct a dedicated study to assess the suitability of Neural MetaLabeler for multi-labeling problems. To this end, we will conduct experiments on datasets of different nature (text, images,...). By keeping the number of samples used for training and the complexity of the underlying neural network small enough to allow fast training, a thorough exploration of the hyperparameter space will be possible. As underlying neural networks, we will again experiment with different types. We believe that the success of the content-based Neural MetaLabeler may depend largely on the type of neural network it is employed with. We think so because MLP, CNN, and LSTM all have very different ways to encode the input, for example, text. While the former is word order agnostic (except for n-grams), the CNN and LSTM do encode more word order information.

OverEager Long Short-Term Memory Networks. In our experiments, OE-LSTMs were surprisingly effective on titles. On full-texts, however, the method did not perform well. The reason is that OE-LSTMs always iterate until the maximum sequence length, which most of the full-texts reach anyway. In order to achieve a fair comparison, the implementation of OE-LSTMs needs to be modified, but has to retain the same idea.

The idea behind OE-LSTMs is that after consuming the entire sequence, the LSTM can make additional iterations through its recurrent connections to further refine its memory cell state and output. In the unrolled graph of the LSTM, this corresponds to propagating through several additional layers with tied weights (compare [27, chapter 10.1]). For instance, if the length of a text is 10, but the maximum length of the sequence is 20, the LSTM applies the weights at the recurrent connection 10 times over before producing the final output, and is therefore "deeper" than an ordinary LSTM.

For fairness and consistency, we would like every text to propagate through the same number of extra iterations, regardless of the text's length. This can be easily achieved by appending a padding token a fixed number of times instead of until the maximum sequence length (as we did in this study).

In a future study, we would like to modify the OE-LSTMs as described, thoroughly study their behavior and effect, and classify them within the family of RNN variants (see [27, chapter 10.2, 10.9] for an overview). Moreover, in case of promising results for the text classification task, we will employ them to other text prediction tasks as well.

E.3 Techniques for Large Sample Sizes

In this study, our goal was to make use of the large amounts of labeled titles by employing complex models that have a high capacity. Moreover, we developed Neural MetaLabeler in an attempt to leverage the surplus of samples in titles to predict the number of labels to be assigned to a publication more accurately than on full-texts.

Several further techniques come to mind that could potentially also benefit from large datasets in particular. First, multi-labeling techniques other than MultiLabeler could be examined. Secondly, one could incorporate co-occurrence information about the labels. Lastly, one could train very large ensembles of neural networks,

or augment them with the nearest-neighbor algorithm. In a future study, we would like to conduct more experiments on titles and full-texts with one or several of the following techniques.

Multi-labeling Techniques. Solutions that employ smart multi-labeling techniques benefit from a large number of labeled data twice. First, like every other classifier, they produce better confidence scores for each label. Secondly, the label selection has many more examples to learn which of the labels to finally select. For example, Nam et al. [61] learn an assignment threshold via linear regression. Chen et al. [14] use an RNN to predict a sequence of labels from the output of a CNN, which in turn operates on the original text.

Label Co-occurence Information. Kurata et al. [45] and Baker et al. [4] proposed similar approaches to leveraging label co-occurrence information for improving the multi-label classification performance via a specialized weight initialization scheme. Here, the weight matrix between the final hidden layer and the output layer is initialized in such a way that each row corresponds to a label co-occurrence pattern. In the respective row, the columns of the corresponding co-occurring labels are set to fixed values, while all others are either initialized randomly or set to zero. Their approaches differ in what counts as a co-occurrence of labels. While Kurata et al. [45] only regard co-occurrence in the gold-standard, Baker et al. [4] work with hierarchical label taxonomies and therefore take hypernyms into account. Both papers find the initialization strategy to improve the classification performance.

When the number of samples in a dataset and consequently the amount of information on label co-occurrence is as large as in our study, the beneficial effect of such an initialization strategy may be more prominent. However, it is important to note that all experiments in these two studies are conducted on datasets that are relatively small in terms of labels and samples. Consequently, there are also a small number of label co-occurrence patterns. In both papers, it is noted that the number of different co-occurrence patterns does not exceed the number of units in the hidden layer (which corresponds to the number of rows in the matrix to initialize). In the very large datasets we considered in our study, already the number of labels exceeds the hidden layer size, let alone the number of co-occurrence patterns. In order to utilize such an initialization scheme for larger datasets, one needs to find some way of reducing the number of co-occurrence patterns. The authors propose to limit the number of patterns to the k most frequent ones. However, this would discard much of the label dependency information. A different approach could be to group similar labels together, for instance through clustering or by leveraging relations in a (hierarchical) thesaurus.

Finally, it should be noted that the initialization scheme obtained from examining the label co-occurrences in a possibly large title dataset could also be used in a full-text-based neural network.

Ensembles of Neural Networks. Because the types of neural networks explored in this study are of different nature, they are also likely to make different errors, and are thus highly uncorrelated. Therefore, it might be very beneficial to explore ensembles of these methods. A typical approach is to train each individual network separately on the same training data and aggregate their outputs in a majority voting or by stacking another classifier on top [49, 103]. A different approach is to combine the neural networks into a single model (for instance, by concatenating their last hidden layers) and to train them jointly with the same loss function (also *coupled ensembles* [20]). The former approach has been widely used in different applications [103] and has been shown to improve the performance if each individual classifier is better than random [31]. On the other hand, coupled ensembles may be particularly suited for our use case because the demand for a large sample size introduced by training multiple large neural networks jointly can be satisfied.

Instance-level Information for Neural Networks. Recently, Wang et al. [90] introduced a neural network architecture that incorporates instance-level information from the training set by considering the text encoded by a bidirectional LSTM and the labels of the k nearest neighbors for prediction as well. In their study, the addition of these information improves the classification performance of a bidirectional LSTM in all cases. However, by replacing the text encoder in their framework, it can easily be generalized to other types of neural networks such as the CNN and MLP in our study, as well. If in a future work we are able to control its computational cost, the k-nearest-neighbor algorithm is particularly attractive for our study due to its well-known consistency properties in the limit with respect to the performance [18].

F IMPLEMENTATION OVERVIEW

In this section, we provide an overview of the implementation of the technical components of this study. The evaluation framework is based on the open-source multi-labeling framework "Quadflor". For the implementation of the neural networks, the functionality of the TensorFlow library was used, which we wrapped as a scikit-learn estimator for compatibility with Quadflor.

F.1 Quadflor

Quadflor[18] is a Python3 framework for multi-label text classification published by Galke et al. [24]. It provides a generic pipeline for multi-label text classification that in turn is based on the scikit-learn[19] framework.

In scikit-learn, a `sklearn.pipeline.Pipeline` is composed of several "Transformers", which transform the input into some form that is interpretable by an "Estimator", which is the last stage of the pipeline. Transformers are classes that inherit from `sklearn.base.TransformerMixin` and implement the `fit(X, y = None)` and `transform(X)` methods. Estimators inherit from `sklearn.base.BaseEstimator` and implement the methods `fit(X, y = None)` and `predict(X)`. We integrated our neural networks into the Quadflor framework by writing a transformer class that transforms a raw text into a sequence of token ids, and by writing an estimator class `MultiLabelSKFlow` to implement the neural networks. The latter is described in the following.

[18] https://github.com/quadflor/Quadflor
[19] http://scikit-learn.org

F.2 TensorFlow

TensorFlow[20] is an open-source deep learning framework that allows to define a neural network as a computation graph. By defining an appropriate node in the graph that defines the loss-function, the library is able to minimize the loss by automatically computing the gradients of learnable parameters in the computation graph. Subsequently, the loss can be minimized using one of the preimplemented optimizers.

TensorFlow exists alongside many other deep learning frameworks that would all provide sufficient functionality to implement our study comfortably. Among those, we chose TensorFlow because it has the largest following and community support, due to which we expected problems to be resolved more quickly than with other frameworks.

F.3 MultiLabelSKFlow

In order to incorporate TensorFlow models into the Quadflor framework, a respective classifier needs to adhere to the fit/predict convention of scikit-learn. The TensorFlow library provides such a wrapper class already (`tf.contrib.learn.SKCompat`). Unfortunately, at the time of implementation (TensorFlow v.1.4), it did not accept the ground truth to be given as an indicator matrix, which is used in scikit-learn/Quadflor for multi-label problems. Hence, using that wrapper class was not an option. As a different option, the `tensorflow.contrib.learn` API also provides an interface that allows for easy implementation of neural network architectures (`tf.contrib.learn.Estimator`). This interface abstracts from the tedious `tf.Session` management and optimization process. However, it is also less flexible than the low-level API of TensorFlow. In particular, we found mini-batch training inconvenient to implement and also computationally slow because it requires initializing a new session for each mini-batch. Furthermore, the built-in monitoring functionalities are marked as deprecated. For these reasons, we implemented the `MultiLabelSKFlow` class to wrap the low-level TensorFlow functionality.

The `MultiLabelSKFlow` class mirrors the structure of what is described in Section 3. As input, besides hyperparameters that control the training procedure such as learning rate, batch size, etc., it expects a function `get_model()` that is supposed to return a TensorFlow computation graph that describes a neural network up to the last hidden layer. The output layer as well as the training procedure is managed by the `MultiLabelSKFlow` class. In order to implement a new neural network model, a user only has to define a `get_model()` function, which is described next. Subsequently, we briefly describe how the main functionality of the neural networks were implemented.

The get_model() function. An implementation of the `get_model(X, y)` function has to take as input the samples `X` and the gold-standard `y`, where `X` is an array or a sparse matrix, and `y` is an indicator matrix. The input parameters are usually used to infer the number of connections between the input features and the hidden layers of the network, or to create a `tf.placeholder` that is used by TensorFlow to propagate data through the computation graph.

When calling the `get_model()` function, the `MultiLabelSKFlow` class uses the function's returned values to build the final neural network and commence training. The function has to return a number of components that are listed in order of expected return in Table 27.

MLP. MLPs with dropout can be implemented using the `tf.contrib.layers.fully_connected` operation followed by `tf.nn.dropout`. `tf.contrib.layers.fully_connected` accepts as parameter an operation to be used as activation function (e.g., `tf.nn.relu` for the rectifier activation function). For Self-Normalizing Neural Networks, Klambauer et al. [44] provide an open-source implementation[21] for alpha-dropout and SELUs, which we adopted for our implementation. `tf.contrib.layers.fully_connected` also accepts a function for normalization as parameter, where we passed `tf.layers.batch_normalization` to implement Batch Normalization.

LSTM. As an implementation of the "vanilla" LSTM formally described in Section B.2.3, the TensorFlow library provides the component `tf.contrib.rnn.LSTMCell`. For unrolling and proper handling of the sequence length, `tf.nn.dynamic_rnn` and `tf.nn.bidirectional_dynamic_rnn` provide the functionality for unidirectional RNNs and bidirectional RNNs, respectively. By passing the maximum sequence length instead of the true sequence length as parameter to these functions, we implemented OE-LSTMs.

The attention mechanism by Yang et al. [93] can be implemented by applying the sequence of mathematical operations that are formally described as equations in Section B.2.3. The respective Python code is shown in Figure 4.

CNN. When sliding a window over the text, we apply a one-dimensional convolution operation. TensorFlow v.1.4 only provides a two-dimensional convolution operation `tf.nn.conv2d`, however, by specifying the height of the input as 1, the operation effectively applies a one-dimensional convolution.

We implemented dynamic max-pooling by creating a mask for each chunk such that all outputs from the detector stage that do not belong to the chunk are set to zero. Afterwards, we apply regular max-pooling (`tf.nn.max_pool`) over the masked output of the detector stage.

Since the bottleneck layer is simply another fully-connected layer, we again used `tf.contrib.layers.fully_connected` for implementation.

[20] https://www.tensorflow.org/

[21] https://github.com/bioinf-jku/SNNs

Component	Python Type	Functionality
x_tensor	tf.placeholder	Used by `MultiLabelSKFlow` to pass input data to the model at training and test time.
y_tensor	tf.placeholder	Used by `MultiLabelSKFlow` to pass the ground truth to the model during training.
last_layer	tf.Tensor	The TensorFlow computation graph from input layer to last hidden layer of the implemented neural network.
params_fit	dictionary	Parameters to be added to the feed dictionary for training (e.g., {keep_probability_placeholder → p})
params_fit	dictionary	Parameters to be added to the feed dictionary at prediction time (e.g., {keep_probability_placeholder → 1.0})
initializer_operations	list of (tf.Tensor, dictionary)	A list of pairs consisting of operations for initializing variables (e.g., embedding tables) before training starts, and the feed dictionary with data to execute the initialize operation.

Table 27: Components that have to be returned by `get_model()`.

```
# we perform attention according to Hierarchical Attention Network (word attention)
## compute hidden representation of outputs
### context vector size as in HN-ATT
context_vector_size = 100
hidden_output_representation = tf.contrib.layers.fully_connected(output_state,
    context_vector_size, activation_fn = tf.nn.tanh)
## compute dot product with context vector
context_vector = tf.Variable(tf.random_normal([context_vector_size], stddev=0.1))
dot_product = tf.tensordot(hidden_output_representation, context_vector, [[2], [0]])
## compute weighted sum
attention_weights = tf.nn.softmax(dot_product)
attention_weights = tf.expand_dims(attention_weights, -1)
output_state = tf.reduce_sum(tf.multiply(output_state, attention_weights), axis = 1)
```

Figure 4: Python code to implement the attention mechanism from Yang et al. [93]. The variable `output_state` is returned by `tf.nn.dynamic_rnn`.

REFERENCES

[1] Ashraf H Abdelwahab, Ahmed Z Emam, Hokey Min, and Adel S Elmaghraby. 2000. Effect of sampling size on Data Mining using Artificial Neural Networks. In *Proceedings of the 10th Conference on Intelligent Engineering Systems Through Artificial Neural Networks*.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[3] Rohit Babbar and Bernhard Schölkopf. 2017. DiSMEC: distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 721–729.

[4] Simon Baker and Anna Korhonen. 2017. Initializing neural networks for hierarchical multi-label text classification. In *BioNLP 2017, Vancouver, Canada, August 4, 2017*. 307–315.

[5] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[7] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[8] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.

[9] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*. 730–738.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[11] Lutz Bornmann and Rüdiger Mutz. 2015. Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references. *Journal of the Association for Information Science and Technology* 66, 11 (2015), 2215–2222.

[12] Damien Brain and G Webb. 1999. On the effect of data set size on bias and variance in classification learning. In *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*. 117–128.

[13] Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Yuchen Wu. 2012. Sample size selection in optimization methods for machine learning. *Mathematical programming* 134, 1 (2012), 127–155.

[14] Guibin Chen, Deheng Ye, Zhenchang Xing, Jieshan Chen, and Erik Cambria. 2017. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2377–2383.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[16] Donald B. Cleveland and Ana D. Cleveland. 1990. *Introduction to indexing and abstracting (2. ed.)*. Libraries Unlimited.

[17] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2017. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Vol. 1. 1107–1116.

[18] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[19] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. 2017. Sharp Minima Can Generalize For Deep Nets. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 1019–1028.

[20] Anuvabh Dutt, Denis Pellerin, and Georges Quénot. 2017. Coupled ensembles of neural networks. *arXiv preprint arXiv:1709.06053* (2017).

[21] Ronen Eldan and Ohad Shamir. 2016. The power of depth for feedforward neural networks. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016,*

New York, USA, June 23-26, 2016. 907–940.

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

[23] Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 1019–1027. http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks

[24] Lukas Galke, Florian Mai, Alan Schelten, Dennis Brunsch, and Ansgar Scherp. 2017. Using Titles vs. Full-text as Source for Automated Semantic Document Annotation. In *Proceedings of Knowledge Capture, Austin, Texas, United States, December 4th-6th*. 9.

[25] Lukas Galke, Ahmed Saleh, and Ansgar Scherp. 2017. Word Embeddings for Practical Information Retrieval. In *47. Jahrestagung der Gesellschaft für Informatik, Informatik 2017, Chemnitz, Germany, September 25-29, 2017*. 2155–2167.

[26] Roberto Gatta, Mauro Vallati, Berardino De Bari, and Mahmut Ozsahin. 2014. The impact of different training sets on medical documents classification. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Assistive Medicine-Volume 1213*. CEUR-WS. org, 1–5.

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

[28] Edouard Grave, Tomas Mikolov, Armand Joulin, and Piotr Bojanowski. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*. 427–431.

[29] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* (2017).

[30] Gregor Große-Bölting, Chifumi Nishioka, and Ansgar Scherp. 2015. A Comparison of Different Strategies for Automated Semantic Document Annotation. In *Proceedings of the 8th International Conference on Knowledge Capture, K-CAP 2015, Palisades, NY, USA, October 7-10, 2015*. ACM, 8:1–8:8.

[31] Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence* 12, 10 (1990), 993–1001.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[33] Bradley M Hemminger, Billy Saelim, Patrick F Sullivan, and Todd J Vision. 2007. Comparison of full-text searching to metadata searching for genes in two biomedical literature cohorts. *Journal of the Association for Information Science and Technology* 58, 14 (2007), 2341–2352.

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

[35] Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 1729–1739.

[36] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[37] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366.

[38] Minlie Huang, Aurélie Névéol, and Zhiyong Lu. 2011. Recommending MeSH terms for annotating biomedical articles. *Journal of the American Medical Informatics Association* 18, 5 (2011), 660–667.

[39] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 448–456. http://jmlr.org/proceedings/papers/v37/ioffe15.html

[40] Thorsten Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the 10th European Conference on Machine Learning*. Springer-Verlag, 137–142.

[41] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[42] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1746–1751. http://aclweb.org/anthology/D/D14/D14-1181.pdf

[43] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[44] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-Normalizing Neural Networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 972–981.

[45] Gakuto Kurata, Bing Xiang, and Bowen Zhou. 2016. Improved Neural Network-based Multi-label Classification with Better Initialization Leveraging Label Co-occurrence. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. 521–526.

[46] Hoa T Le, Christophe Cerisara, and Alexandre Denis. 2018. Do Convolutional Networks need to be Deep for Text Classification?. In *Proceedings of the AAAI-18 Workshop on Affective Content Analysis, New Orleans, Louisiana, United States, February 3rd*.

[47] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[48] Ladislav Lenc and Pavel Král. 2016. Deep neural networks for Czech multi-label document classification. In *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2016), Konya, Turkey (3-9 April 2016)*.

[49] Ladislav Lenc and Pavel Král. 2017. Ensemble of Neural Networks for Multi-label Document Classification. In *Proceedings of the 17th conference ITAT 2017. Martinske hole, Slovakia, September 22-26, 2017*. 186–192.

[50] Ladislav Lenc and Pavel Král. 2017. Two-Level Neural Network for Multi-label Document Classification. In *Artificial Neural Networks and Machine Learning - ICANN 2017 - 26th International Conference on Artificial Neural Networks, Alghero, Italy, September 11-14, 2017, Proceedings, Part II*. 368–375.

[51] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. 115–124.

[52] Ke Liu, Shengwen Peng, Junqiu Wu, Chengxiang Zhai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2015. MeSHLabeler: improving the accuracy of large-scale MeSH indexing by integrating diverse evidence. *Bioinformatics* 31, 12 (2015), i339–i347.

[53] Eneldo Loza Mencía and Johannes Fürnkranz. 2010. Efficient Multilabel Classification Algorithms for Large-Scale Problems in the Legal Domain. In *Semantic Processing of Legal Texts: Where the Language of Law Meets the Law of Language*. 192–215.

[54] Yuqing Mao and Zhiyong Lu. 2017. MeSH Now: automatic MeSH indexing at PubMed scale via learning to rank. *Journal of biomedical semantics* 8, 1 (2017), 15.

[55] Olena Medelyan and Ian H. Witten. 2008. Domain-independent automatic keyphrase indexing with small training sets. *Journal of the Association for Information Science and Technology* 59, 7 (2008), 1026–1040.

[56] Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589* (2017).

[57] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 3111–3119.

[58] James G. Mork, Dina Demner-Fushman, Susan Schmidt, and Alan R. Aronson. 2014. Recent Enhancements to the NLM Medical Text Indexer. In *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014*. 1328–1336.

[59] James G. Mork, Antonio Jimeno-Yepes, and Alan R. Aronson. 2013. The NLM Medical Text Indexer System for Indexing Biomedical Literature. In *Proceedings of the first Workshop on Bio-Medical Semantic Indexing and Question Answering, a Post-Conference Workshop of Conference and Labs of the Evaluation Forum 2013 (CLEF 2013) , Valencia, Spain, September 27th, 2013*.

[60] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. 807–814.

[61] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. 2014. Large-Scale Multi-label Text Classification - Revisiting Neural Networks. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*. 437–452.

[62] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J. Kim, and Johannes Fürnkranz. 2015. Predicting Unseen Labels Using Label Hierarchies in Large-Scale Multi-label Learning. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I*. 102–118.

[63] Cristiano Nascimento, Alberto H. F. Laender, Altigran Soares da Silva, and Marcos André Gonçalves. 2011. A source independent framework for research paper recommendation. In *Proceedings of the 2011 Joint International Conference on Digital Libraries, JCDL 2011, Ottawa, ON, Canada, June 13-17, 2011*. 297–306.

[64] Chifumi Nishioka and Ansgar Scherp. 2016. Profiling vs. Time vs. Content: What does Matter for Top-k Publication Recommendation based on Twitter Profiles?. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries, JCDL 2016, Newark, NJ, USA, June 19 - 23, 2016*. 171–180.

[65] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International*

Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013. 1310–1318.

[66] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1532–1543.

[67] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. 701–710.

[68] Bruno Pouliquen, Ralf Steinberger, and Camelia Ignat. 2003. Automatic annotation of multilingual text collections with a conceptual thesaurus. In *Proceedings of the Workshop 'Ontologies and Information Extraction' at the Summer School 'The Semantic Web and Language Technology - Its Potential and Practicalities' (EUROLAN'2003), Bucharest, Romania, 28 July - 8 August 2003*. 9–28.

[69] Yashoteja Prabhu and Manik Varma. 2014. FastXML: a fast, accurate and stable tree-classifier for extreme multi-label learning. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. 263–272.

[70] Lutz Prechelt. 1998. Early stopping-but when? *Neural Networks: Tricks of the trade* (1998), 553–553.

[71] Nils Reimers and Iryna Gurevych. 2017. Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks. *arXiv preprint arXiv:1707.06799* (2017).

[72] Anthony Rios and Ramakanth Kavuluru. 2015. Convolutional neural networks for biomedical text classification: application in indexing biomedical articles. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics, BCB 2015, Atlanta, GA, USA, September 9-12, 2015*. 258–267.

[73] Timothy N Rubin, America Chambers, Padhraic Smyth, and Mark Steyvers. 2012. Statistical topic models for multi-label document classification. *Machine learning* 88, 1 (2012), 157–208.

[74] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[75] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[76] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533.

[77] Tim Salimans and Diederik P. Kingma. 2016. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 901.

[78] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[79] Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. 298–307.

[80] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[81] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2960–2968.

[82] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[83] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. 1556–1566.

[84] Lei Tang, Suju Rajan, and Vijay K. Narayanan. 2009. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*. 211–220.

[85] Matus Telgarsky. 2016. benefits of depth in neural networks. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*. 1517–1539.

[86] Misha Teplitskiy, Grace Lu, and Eamon Duede. 2017. Amplifying the impact of open access: Wikipedia and the diffusion of science. *Journal of the Association for Information Science and Technology* 68, 9 (2017), 2116–2127.

[87] Martin Toepfer and Christin Seifert. 2017. Descriptor-Invariant Fusion Architectures for Automatic Subject Indexing. In *2017 ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017, Toronto, ON, Canada, June 19-23, 2017*. 31–40.

[88] George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artieres, Axel Ngonga, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Paliouras. 2015. An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics* 16 (2015), 138.

[89] Princeton University. 2010. About WordNet. https://wordnet.princeton.edu/man/morphy.7WN.html. (2010). Accessed: May 12, 2016.

[90] Zhiguo Wang, Wael Hamza, and Linfeng Song. 2017. k-Nearest Neighbor Augmented Neural Networks for Text Classification. *arXiv preprint arXiv:1708.07863* (2017).

[91] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. 2017. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *arXiv preprint arXiv:1705.08292* (2017).

[92] Yijun Xiao and Kyunghyun Cho. 2016. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367* (2016).

[93] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. 1480–1489.

[94] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv preprint arXiv:1702.01923* (2017).

[95] Wenpeng Yin and Hinrich Schütze. 2015. Multichannel Variable-Size Convolution for Sentence Classification. In *Proceedings of the 19th Conference on Computational Natural Language Learning, CoNLL 2015, Beijing, China, July 30-31, 2015*. 204–214.

[96] Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898* (2017).

[97] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2017. Recent trends in deep learning based natural language processing. *arXiv:1708.02709* (2017).

[98] Min-Ling Zhang and Zhi-Hua Zhou. 2006. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering* 18, 10 (2006), 1338–1351.

[99] Rui Zhang, Honglak Lee, and Dragomir R. Radev. 2016. Dependency Sensitive Convolutional Neural Networks for Modeling Sentences and Documents. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. 1512–1521.

[100] Wenjie Zhang, Liwei Wang, Junchi Yan, Xiangfeng Wang, and Hongyuan Zha. 2017. Deep Extreme Multi-label Learning. *arXiv preprint arXiv:1704.03718* (2017).

[101] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 649–657.

[102] Ye Zhang and Byron C. Wallace. 2017. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing, IJCNLP 2017, Taipei, Taiwan, November 27 - December 1, 2017 - Volume 1: Long Papers*. 253–263.

[103] Ying Zhao, Jun Gao, and Xuezhi Yang. 2005. A survey of neural network ensembles. In *Proceedings of International Conference on Neural Networks and Brain, 2005. ICNN&B'05*. 438–442.