UNIVERSITY OF TWENTE

MASTER THESIS

HIERARCHICAL DEEP NEURAL NETWORKS FOR MeSH SUBJECT PREDICTION

Author:

Ashwin Sadananda Bhat

Supervisors:

Dr. Gwenn Englebienne

Dr. Mannes Poel

Dr. Shenghui Wang

Rob Koopman

August 26, 2019

UNIVERSITY OF TWENTE.

OCLC™
The world's libraries. Connected.

ACKNOWLEDGEMENTS

Researching the relatively young field of Extreme Multi-Label Classification was a challenging and rewarding experience for me. Without the timely guidance, advice, and patience from my supervisors, Gwenn Englebienne and Mannes Poel at the University of Twente, as well as Rob Koopman and Shenghui Wang at OCLC, this project would not have been possible. I cannot thank Gwenn and Mannes enough for giving me the opportunity to work on this research problem, especially Gwenn for his continuous guidance and taking time out of his busy schedule weekly for follow-ups. I also have to thank Rob and Shenghui for being patient and supportive in the face of mistakes and enthusiastic in the face of results, whether good or bad. I would also like to thank the BOZ at the University of Twente for helping me navigate the graduation process, and to the wonderful team at OCLC for accommodating me during my thesis and making sure that I had the necessary tools to complete my research without any issues or delays.

Student Details

Student Name : Ashwin Sadananda Bhat
Student Number : s1995790

ABSTRACT

Extreme Mutli-Label Text classification (XMTC) problems attempt to assign a few relevant labels to text from an extremely large label-space. XMTC label spaces generally follow a power law distribution, resulting in data sparsity issues for tail labels and aggressive prediction of head labels. Deep learning methods for tackling such large scale problems have recently gained attention and have reached state-of-the-art performance. Notably, XML-CNN[32] is a deep learning architecture that was tailored specifically towards XMTC problems. Assigning relevant labels to medical journals in the Medline dataset is an XMTC problem with a highly skewed label-space and highly arcane terms. This project explored modifications to XML-CNN by implementing a hierarchical XML-CNN architecture to leverage the inter-label relationships for training and classification. An automated hierarchy generated by Hierarchical Agglomerative Clustering and the expert-curated MeSH hierarchy for medline were used to evaluate prediction performance. Borrowing from the concept of multi-task learning, the hierarchies were used to modify the XML-CNN architecture to function as a single model using hard parameter sharing with separate loss functions for each level of the hierarchy. The experiments were focused on testing the effect of the hierarchical approach, the effect of an automatically generated hierarchy and that of a manually curated hierarchy. The use of hierarchies were found to be less suited for medline label prediction than the original XML-CNN model. However, the performance of the hierarchical models were comparable to XML-CNN and is sufficiently high that the hierarchical models cannot be considered ineffective for subject prediction tasks.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

XMC : Extreme Multi-label Classification
XMTC : Extreme Multi-label Text Classification
MTL : Multi-Task Learning
HAC : Hierarchical Agglomerative Clustering
MeSH : Medical Subject Headings
NLM : National Library of Medicine
SVM : Support Vector Machine
DiSMEC : Distributed Sparse Machines for Extreme Multi-label Classification
SLEEC : Sparse Local Embeddings for Extreme Multi-label Classification
SVD : Singular Value Decomposition
kNN : k Nearest Neighbours
nDCG : Normalised Discounted Cumulative Gain
CNN : Convolutional Neural Networks
XML-CNN : eXtreme Multi-Label CNN
NCBI : National Center for Biotechnology Information
OCLC : Online Computer Library Center
BCE : Binary Cross Entropy
GPU : Graphics Processing Unit
CPU : Central Processing Unit
IDE : Integrated Development Environment
NLP : Natural Language Processing
TF-IDF : Term Frequency-Inverse Document Frequency

1 INTRODUCTION

1.1 Problem Statement

Supervised classification problems in machine learning involve identifying the 'class' of a particular item (such as an image, a document, an audio or video clip, etc.), for example, identifying whether a given picture contains a dog or a cat. Such problems usually involve a dataset consisting of various labelled examples of the items being classified (such as a series of labelled pictures of cats and dogs) which is used to train classifiers to learn from these labelled examples and correctly identify the classes of previously unseen examples. Extreme Multi-Label Classification (XMC) is a supervised machine learning problem of assigning an item a relevant subset of labels chosen from an extremely large set of labels. Such problems are often seen in real-world domains such as product categorisation in e-commerce [2][8][47], online ad-recommendation[42] [40], hash-tag recommendation in social media platforms [18], etc. XMC differs from multi-class classification and multi-label classification. Traditional multi-class classification seeks to predict a single label from a set of mutually exclusive labels (i.e., one item should have one and only one label associated with it), and multi-label classification seeks to predict all relevant labels from a relatively small set of labels that are not mutually exclusive. XMC is most similar to multi-label classification with the caveat that the label spaces in XMC-style problems can reach gigantic proportions. XMC is a unique problem that can best be described as a multi-class, multi-label problem with a label space that can range from several thousands to millions of labels per data point. When the application domain is text, it is called Extreme Multi-Label Text Classification (XMTC). For example, assigning relevant tags to Wikipedia articles falls under the problem of XMTC, where a handful of labels need to be assigned to each article, chosen from a label-space of over a million unique labels. Extreme classification come with challenges such as scalability issues due to the extremely large label spaces, data sparsity issues stemming from insufficient training samples for infrequently appearing labels, severe class imbalance between labels, an incomplete ground truth of labels, etc.

There has been considerable research conducted in the XMC field, with varying approaches such as Label Embedding Methods, Tree-based methods, Deep Learning methods, etc[9][42][5][31][22] [50][32]. The application of deep learning methods to XMTC problems has gained significant attention in recent years due to the success of deep learning in other areas such as computer vision, machine translation, language modelling, etc. Recent work shows promising results for deep learning in the XMTC domain[32][57][48]. However, the application of deep learning to XMTC is a relatively new field with a lot of potential.

1.2 Goal

This project was undertaken in order to examine the potential of applying deep learning methods using Multi-Task-Learning (MTL) to the XMTC problem of subject prediction for medical journals in the Medline dataset[37]. Specifically, deep learning architectures with hard parameter sharing to automatically assigning relevant medical subject headings (MeSH terms) to research papers in the Medline dataset based on their abstracts and meta-data by leveraging the relationships between the labels. This falls under the XMTC domain and is complicated by the fact that the label space contains terms that are specific to the medical domain and are unlikely to be found in other vocabularies. Additionally, labels in the XMC field often contain some semantic inter-relationships, i.e., labels used to identify a particular item will often be semantically related to each other in context. For example, a medical journal that describes advancements in the field of brain cancer treatments with experiments on mice may be annotated with labels such as 'cancer', 'glioblastoma', 'treatment', 'mice', etc (among others). It is reasonable to expect that these terms have a semantic relationship with each other in the context of the text, and that some labels may be more specific

to and descriptive of the given example than others (such as 'glioblastoma' being more descriptive than simply 'cancer'). As such, these labels can be expressed as a hierarchy, where similar3 labels are clustered together, the generic labels are near the top (root) of the hierarchy, and the specific labels at the bottom (leaf). Hierarchies have been shown to make it easier to learn from a moderately sizeable number of labels[10][44][12] and can also be exploited to learn classifiers that perform well under the extreme classification setting [40] [2].

Hierarchies can be automatically generated or manually created. It is worthwhile to compare the performance of automatically generated hierarchies to manually created hierarchies. Most large-scale extreme classification applications do not have any manual hierarchies associated with them, and hence automatically generated hierarchies are often the only choice. However, in some rare cases, some datasets have manually maintained hierarchies that can be leveraged. In the case of this project, the dataset used has a manually created label-hierarchy associated with it, called the MeSH hierarchy. An automatic hierarchy, generated through Hierarchical Agglomerative Clustering, is also used in the experiments. These hierarchies are used to modify a state-of-the-art deep learning method for extreme classification, named XML-CNN. The existing architecture of XML-CNN is modified by grouping the output nodes of the model based on their position in the hierarchy and treating each group as a separate task, each with it's own loss function. This approach borrows from Multi-Task Learning (MTL)[11], where one network is used to train for multiple tasks at once. In the case of XMTC, the task of predicting the more frequent and general terms can be distinguished from that of predicting the more specific and infrequent terms. The intuition is that the hierarchy will separate the more general terms from the specific terms and the hidden layer representation of the specific terms will be bolstered by the general terms through shared parameters. This idea is discussed in detail in section 4.2. The motivating research questions for this project are defined as:

(1) **RQ 1:** *How does the inclusion of the label hierarchy for multi-task learning affect the performance of the model?*

(2) **RQ 2:** *What impact does an artificially generated hierarchy have on the performance?*

(3) **RQ 3:** *How does the performance vary when a manually created hierarchy is used over an artificially generated hierarchy?*

The structure of this report is as follows: Section 2 details the background for this project, including the social relevance of the project, common issues in the XMTC domain, and the related work in extreme classification which details the different types of approaches used to tackle the problem in literature and some of the state-of-the-art models in each type of approach. Section 3 describes the Medline dataset, detailing the features of the dataset as well as the characteristics of the MeSH terms. Section 4 details the important aspects of the experiment method such as the architecture and construction of the baseline model and the hierarchical models, evaluation metrics, environment setup, working with extreme multi-label datasets, etc. Sections 5 and 6 discusses the results and insights obtained from experiments. Lastly, section 7 concludes the report and offers remarks for future work. The appendix A contains supplementary information such as experiment parameters.

2 BACKGROUND

2.1 Social Relevance

Since the first academic journal, *Journal des sçavans*, was published in 1665, the number of scientific literature being published each year has constantly been on the rise. In 2010, research from the University of Ottawa reported that the total number of research papers had passed 50 million [23]. The 2018 STM report [24] estimates that over three million articles are published each year by seven to eight million authors globally. The growth rate for publications was reported to be around 4% in recent years [24] and could accelerate in the coming years. The average growth rate for medical informatics literature from 1987 to 2006 was 12%[19]. This ever-increasing trend in the number of research articles, while adding to the collective human knowledge, also comes with it's own set of challenges.

One such challenge is that of identifying the subject matter and assigning relevant labels to each document. Traditionally, assignment of labels has been done manually by the authors of the papers and/or trained annotators, either by utilising a controlled vocabulary, free assignment, or a combination of both. Controlled vocabulary systems offer little flexibility, but make it easier for information retrieval systems to function by eliminating the ambiguity associated with free assignment. Some examples of such systems include Canadian Subject Headings (CSH), Polythematic Structured Subject Heading System (PSH), Medical Subject Headings(MeSH), etc. The obvious downside of purely controlled assignment is that it limits authors from assigning terms that may be extremely specific to their research that aren't in the vocabulary.

Free assignment also has it's limitations in quality and relevance of assigned labels, as well as a general lack of standards and structure. For medical journals in Medline, Trained annotators at the National Library of Medicine (NLM) assign an average of thirteen labels per article[43]. If free assignment was used instead of MeSH vocabulary, the quality of these labels could easily be problematic if annotators are not familiar with field of research concerning certain articles, requiring authors, who are by-default experts in their work, to assign labels to their own articles as well. While it is certainly easier for authors to attach relevant labels to their own work (being intimately familiar with the nature and content of their work), it becomes difficult when an author has to assign relevant labels relative to other documents and their labels in a database. This makes it difficult for information retrieval systems to effectively index and organise articles based on their tags. Automating the labels assignment task can save time and, more importantly, ensure that the assigned labels make it easier for digital systems to characterise a document relative to other documents in the database, thereby enabling it to index, organise, and retrieve documents with ease and quickness. Additionally, going through a large label-space to find the most relevant tags is time consuming and quickly becomes an impossible task for humans in the extreme setting. With such a large influx of papers being published annually, the amount of data that will need labelling will only increase. As such, it is evident that there is a lot to be gained from developing reliable and accurate techniques to assign relevant labels to academic papers.

Academics isn't the only field that stands to gain immensely from advances in Extreme Classification ethods. Outside of the academic articles, developing a robust system for automatic label assignment can have an impact across multiple domains. While this project focuses on the Medline dataset, the approach developed can be generalised to any XMTC problem, from subject prediction for research journals in other domains to other applications such as: *Ranked Retrieval for Search Engines*, *Advertising*, *Language Modelling*, *Item-to-Item Recommendations*, *Object Recognition in Computer Vision*, *Gene Function Prediction*, etc. For example, such systems can serve to improve retrieval tasks in large systems such as digital libraries, search engines, video streaming services, etc. Automating the generation of relevant labels for these tasks can significantly reduce the human

effort involved in curating and assigning labels to fit content. XMC solutions can also be adapted to recommendation problems, which are often at the heart of many applications such as targeted advertising, product recommendations for online retail, movie/song recommendation systems such as those used by Netflix and Spotify, etc. With such a wide range of applications, it is easy to argue that exploring methods to address extreme classification problems is a worthwhile pursuit that can save a large amount of human effort in the organisation, retrieval, and recommendation of digital data.

2.2 Related Work

The existing body of work in Extreme Multi-Label Classification can be broadly categorised into the following types of approaches: (1) One-Vs-All, (2) Embedding Based, (3) Tree Based, and (4) Deep Learning. These approaches are described below.

2.2.1 One-Vs-All Methods. In traditional multi-label classification, it is common to use a *one-vs-all* classifier to learn the mapping from input to label via separate classifiers independently for each label such as one-vs-all SVM. This approach is valid for multi-label classification due to the relatively small label space, but rarely translates well to XMTC problems as the extremely large labels spaces and large number of training instances increase computational complexity and model size to an unfeasible level. However, there have been a few approaches that attempt to reduce the complexity. PD-Sparse[56] is one such approach that attempts to reduce complexity by training classifiers for each label. The assumption in this work is that there are only a few correct labels for each instance and that the feature space is rich enough to make the distinction between labels clear. Under this assumption, PD-Sparse uses a margin-maximising loss function in combination with L_1 penalty to achieve extremely sparse solution in extreme classification. An extension of PD-Sparse is the PPD-Sparse[55] method which utilises large-scale distributed computing to efficiently parallelise the PD-Sparse algorithm.

DiSMEC[5] is a state-of-the-art one-vs-all linear classifier method that is a large-scale distributed framework. DiSMEC discards spurious weight coefficients that are extremely close to zero, giving it the ability to keep the model compact in size, without losing prediction accuracy. This makes DiSMEC much smaller in size compared to other state-of-the art models.

2.2.2 Embedding Based Methods. Embedding methods attempt to circumvent the problem of data sparsity and large label spaces by compressing label vectors into low-dimensional representations for training instances. During prediction, these low dimensional representations are decompressed back to the original label space. Consider that the training data is of the form (x_i, y_i), where $i = \{1, 2, ..., n\}$, $x_i \in \mathbb{R}$, $y_i \in \{0, 1\}^L$, D is the total number of features and L is the total number of labels. Then, instead of finding the mapping from a random x to it's relevant y for a very large L, embedding methods attempt to compress label vectors y_i from L-dimension to a lower $\hat{L}$-dimensional embedding vector z_i, given by $f_C(y_i)$ (where f_C is a compression function) by either linear or non-linear projections. Once in a lower dimensional space, classifiers such as SVMs can be used to efficiently find mappings from x_i to relevant z_i. Subsequently, the predicted low dimensional vector z_i is decompressed back to the original L-dimensional space to get the predicted label vector $\hat{y_i}$ as $\hat{y_i} = f_D(z_i)$ (where f_D is a decompression function). Embedding methods vary based on the design of the compression and decompression functions used such as Bloom Filters[15], Singular Value Decomposition (SVD)[51], Compressed Sensing[21][25], etc.

One of the more well-known embedding methods is SLEEC[9], which obtains z_i by preserving the pairwise distance of only the closest label vectors (as opposed to all label vectors) by setting a threshold of some distance metric, d, such that $d(z_i, z_j) \approx d(y_i, y_j)$ if and only if the i^{th} label is a

nearest neighbour of the j^{th} label. During prediction, a kNN classifier is used in the embedding space, which leverages the nearest neighbour relationship that was preserved during training, as opposed to a decompression matrix. Since kNN classifiers have high computational complexity, SLEEC clusters the training data and learns embeddings for each cluster, making it easier to search for nearest neighbours during prediction time by only looking within the cluster to which a new document belongs.

AnnexML[50], that uses graph embeddings, is a new embedding method that is an extension of SLEEC. AnnexML works by generating a graph of similar training samples using kNN, connecting samples that have similar label vectors and partitioning the graph into k sub-graphs while maintaining the original graph structure. Projection matrices are then learned for each sub-graph. During prediction, each new test sample is identified as belonging to a particular partition. The test point is then projected onto that partition and using a kNN classifier, the nearest neighbours within the partition are determined, and the labels belonging to the nearest neighbours are returned as the result. Currently, AnnexML appears to be the state-of-the-art in embedding based method along with SLEEC.

A new semantic embedding approach called Ariadne[27] has been developed recently by OCLC, which has performed very well on subject prediction tasks. The Ariadne method works by using random projections to embed terms to a lower dimension by directly computing the lower dimensional representation of each word as it goes through the corpus, making the approach extremely fast. In traditional language-related tasks, extremely frequent words (such as stop-words) are handled by disregarding them, as they provide little to no discriminatory effect. In the Ariadne method, terms are given continuous weights (inversely proportional to their frequency) and an "average language vector" of the corpus is computed. Intuitively, this vector will represent the most frequent, and therefore, least discriminative words. Word vectors are projected onto the hyperplane orthogonal to the average language vector. In effect, this keeps the influence of words with low discriminative power (most frequent words) to a minimum. For subject prediction tasks, the terms with vectors that are similar to the average vector are down-weighted when creating the document embedding, further increasing the discriminative capabilities of the final embedding, and the nearest subject neighbours of the new document embedding are predicted. This is possible as Ariadne embeds documents and subjects in the same space. This method, while performing well on extreme classification tasks, is also extremely fast and therefore, practical in approach. An implementation of Ariadne for MeSH subject prediction was provided by OCLC for comparisons against the deep learning methods implemented in this project. The performance of this model is also included in the results section along with the results of the models tested as part of this project in section 5.

2.2.3 Tree Based Methods. Tree based methods, as the name suggests, use decision trees for classification. Rather than classic decision trees, these methods make use of an ensemble of decision trees. Recent variants of these methods have improved the state-of-the-art in XMTC problems[42][54][2]. The rationale behind tree based ensemble methods is to recursively partition training instances by features at non-terminal nodes that give relatively simple classifiers at the leaf nodes, each with a small number of active labels. Where classic decision trees select a feature for splitting based on information gain, tree based ensemble methods learn a hyperplane to split an instance at each node, which is equivalent to using a weighted combination of all features. This makes the algorithm less greedy than single-feature based splitting and hence, more robust for extreme classification. Additionally, the training time for such methods is significantly lower.

The most representative method in this category is FastXML[42], which learns a hyperplane and optimises an nDCG-based ranking loss function at each node. The hyperplane splits the training

instances at the current node into two subsets, each containing documents with similar label distributions. During prediction, each new test instance is passed from the root node of each induced tree to the leaf node, and the label distributions in all reached leaves are aggregated. PfastreXML[22] is an extension to FastXML that uses the same architecture, but differs in the selection of the loss function. PfastreXML makes use of propensity scored ranking loss functions such as *propensity scored nDCG* and *propensity scored precision at k*, which specifically attempts to addresses the issue of the missing labels in the ground truth, enabling the algorithm to predict tail labels with higher accuracy than other methods.

Parabel[40] is a relatively newer tree based method which, instead of partitioning training instances, focuses on partitioning labels. Parabel uses the one-vs-all approach in combination with tree-based partitioning to get near state-of-the-art prediction performance while keeping the training time relatively low and with less computational power.

2.2.4 Deep Learning Methods. There have been a relatively fewer deep learning approaches applied in the extreme classification domain. One of the first such methods is XML-CNN[32], which is inspired by CNN-Kim[26]. CNN-Kim, one of the first attempts at using CNNs designed for multi-label classification, uses convolutional layers over concatenated word embeddings of a document to create feature maps that are then max pooled and used by a fully connected layer and an L-dimensional softmax output layer (where L is the number of labels). Although this architecture was applied to multi-label problems, XML-CNN adapted the architecture to better suit the nature of XMTC. The XML-CNN model is used as a baseline for this project and its architecture is described in detail in section 4.1

3 DATASET

The dataset used in this project is the Medline dataset[37], which is a publicly available dataset of medical publications with information such as their abstracts, citations, authors, labels, language of publication, and other meta-data. It contains more than twenty seven million such records and is maintained by the National Center for Biotechnology Information (NCBI) at the U.S National Library of Medicine (NLM) through PubMed, a free online resource that hosts citations and abstracts in the field of medicine. The majority of the articles in medline are bio-medical in their content and the rest pertain to the life sciences. For the purpose of this project, a randomly sampled subset of Medline was used, containing 1 million records and associated meta-data in a text file. Each record is indexed using Medical Subject Headings (MeSH), a controlled vocabulary that is curated and updated yearly by NLM. The main purpose of MeSH terms is to provide an organised set of labels for indexing and cataloguing the vast number of medical journals in the PubMed database for easy retrieval. The MeSH terms are organised in a hierarchical structure and can be accessed through the MeSH Tree View Browser[35].

In the following subsections, 3.1 describes the MeSH terms and details the different types of Medline indexing terms and 3.2 outlines the important characteristics of the Medline subset of 1 million records used in the experiments.

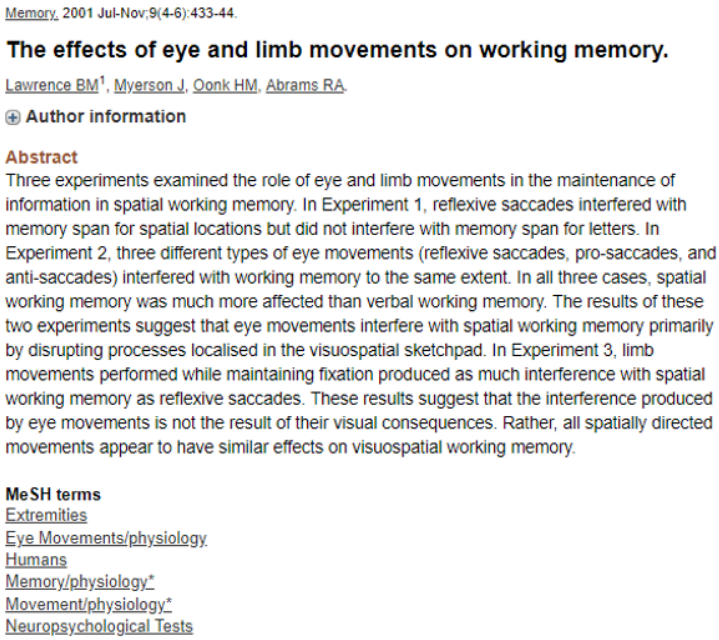
3.1 MeSH Terms

MeSH is a controlled vocabulary of labels that function as keywords to help in the retrieval of documents in the PubMed system. These terms are updated each year, to remain relevant and comprehensive. They are assigned by expert indexers at NLM, who are trained subject specialists in fields such as anatomy, chemistry, etc. The method by which the NLM indexers review journals and articles and assign MeSH terms to articles, as elaborated on the NLM website[36], is based on the importance of each term to bio-medicine and their significance and usage in articles with the goal of highlighting the main concepts and ideas discussed in each article. As a controlled vocabulary, MeSH offers indexers a definitive structure to refer to. However, in the event that no specific terms are available in the vocabulary of MeSH terms for an article, an indexer assigns the closest general term available. Generally, an article may be assigned ten to fifteen terms in order to reasonably cover the concepts within each article.

There are four main types of MeSH terms: *Headings, Subheadings, Publication Types, and Supplementary Concept Records*. MeSH Headings represent concepts that are found in bio-medical articles and can be either broad or specific. They also contain non-clinical topics, depending on the literature.

Additionally, indexers also assign MeSH headings to specify the group being studied. These types of headings are called *check-tags*. These include distinctions such as whether a study was a *human or animal* study, *male or female* study, the *age group* of the study, and the *type* of article. The assignment of check-tags is mandatory and hence, they account for a significant portion of the label distribution in the medline dataset. This can be seen in figure 3, where the most frequent labels are human, animal, male, female, etc., which are very popular, compared to more specific terms like "protein kinases" and "thiazoles". The *type* of the article being indexed is of the ‘Publication Type’ term. These include terms such as ‘Surveys and Questionnaires’, ‘Review’, ‘Practical Guideline’, etc.

Subheadings are specialised terms that, when used in conjunction with headings, provide a more fine-grained description of an article. For example, in figure 1, the subheading "physiology" is used with the headings "Memory", "Movement", and "Eye Movements" to further specify that the article is about the physiology of memory and movement. There are over eighty such subheadings in the MeSH vocabulary.



Memory. 2001 Jul-Nov;9(4-6):433-44.

The effects of eye and limb movements on working memory.

Lawrence BM[1], Myerson J, Oonk HM, Abrams RA.

Author information

Abstract

Three experiments examined the role of eye and limb movements in the maintenance of information in spatial working memory. In Experiment 1, reflexive saccades interfered with memory span for spatial locations but did not interfere with memory span for letters. In Experiment 2, three different types of eye movements (reflexive saccades, pro-saccades, and anti-saccades) interfered with working memory to the same extent. In all three cases, spatial working memory was much more affected than verbal working memory. The results of these two experiments suggest that eye movements interfere with spatial working memory primarily by disrupting processes localised in the visuospatial sketchpad. In Experiment 3, limb movements performed while maintaining fixation produced as much interference with spatial working memory as reflexive saccades. These results suggest that the interference produced by eye movements is not the result of their visual consequences. Rather, all spatially directed movements appear to have similar effects on visuospatial working memory.

MeSH terms

Extremities
Eye Movements/physiology
Humans
Memory/physiology*
Movement/physiology*
Neuropsychological Tests

Fig. 1. Sample Medline Article with related MeSH terms

In addition to the type of MeSH terms described above, there are also 'Supplementary Concepts' terms. These are the terms that provide further clarity to articles on top of other MeSH headings. There are over two hundred thousand Supplementary Concept terms that describe articles on Medline. Most of these terms describe chemical substances mentioned in each article.

The subject headings or labels in the dataset initially consisted of the MeSH headings and subheadings being concatenated and treated as a separate label for each record. During prior experiments, it was found that this approach to treating MeSH headings and subheadings led to most of the labels in the dataset having very few training samples, as the count for the concatenated labels would be far fewer than if the headings and subheadings were treated independently. In this case, the vast majority of the labels appear between one to ten times in the dataset (figure 2). Additionally, the MeSH hierarchy only accounted for the MeSH headings, and not the subheadings, which had no particular structure and were used in conjunction with multiple main headings depending on the context. Due to these reasons, and the fact that the majority of annotated labels in Medline were headings rather than subheadings, the main focus of the experiments was on learning and predicting the main headings.

3.2 Characteristics and Preparation

The labels in medline, similar to other extreme classification domains, follow a power law distribution. This means that the vast majority of the labels are extremely sparse. Even when considering the subject headings alone, the majority are still very sparse for a dataset of 1 million articles. With a few labels dominating in the dataset, it is expected that the model will have a hard time learning meaningful representations for the tail end of the labels. Additionally, these values are for the entire dataset. Once the data had been split into training, validation, and test sets, the training sparsity for the tail end was even more severe than it already was. The distribution of the top thousand most

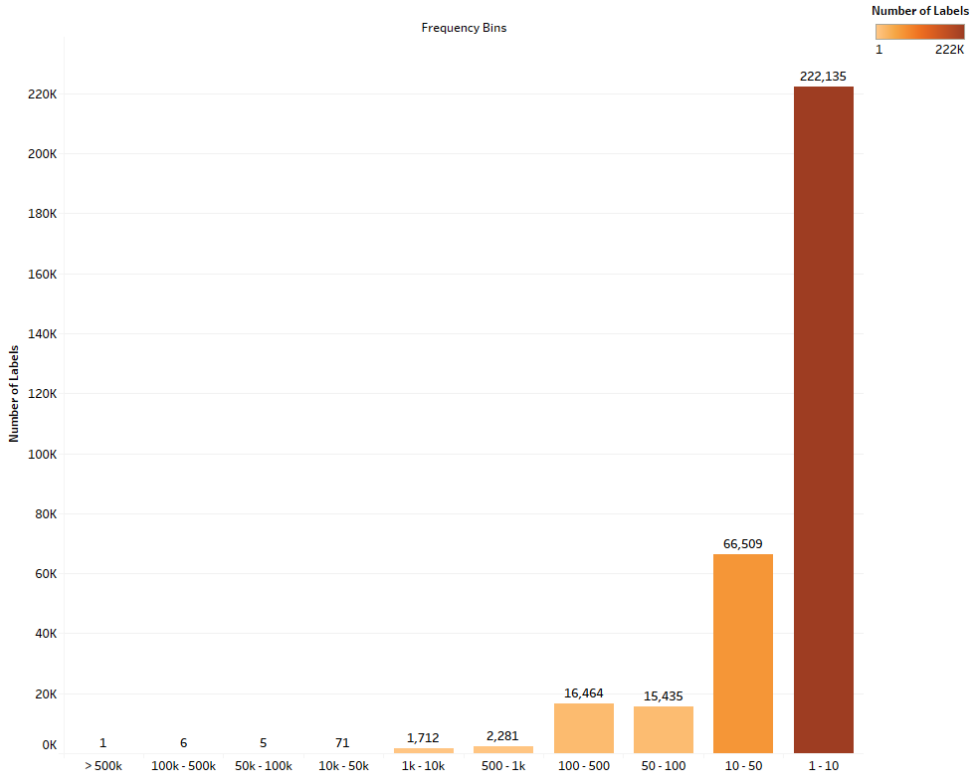


Fig. 2. Number of labels per distribution bin for all MeSH headings and subheadings in the dataset prior to cleaning.

frequent labels in the subset is shown in figure 3. This shows how quickly the available instances of labels drop, leading to severe data sparsity. Due to this sparsity, many tail labels do not offer enough samples for deep learning networks to learn any meaningful information about them with so few instances. Hence, a cut-off limit is set on frequency of labels to filter out those labels that do not appear at least ten times in the training set.

3.2.1 Label Distribution. For the dataset under consideration, the number of MeSH headings in the dataset totalled nearly twenty two thousand. Of these, the labels 'humans' far outstrips other labels in sheer number of occurrences, at nearly 580k. Only seven labels (humans, male, female, animals, adult, middle aged, and aged) appear more than 100k times, with 'humans' being significantly more common than the second most common label 'male' (580k to 330k). Another five labels (adolescent, mice, rats, time factor, and child) appear between 50k and 100k times. The frequency of labels drops sharply from that point on, with seventy-one labels that have a frequency of double-digit-thousands (10k to 50k). The frequency distribution for the top 1k most frequent labels is shown in figure 3. The class imbalance is clearly seen in the frequency distribution, where the head of the distribution is almost completely covered by check-tags (humans, male, female, animals, etc.). This extreme imbalance in label distribution was an indication that the model might have the tendency to learn the label distribution and predict the head labels aggressively and the tail labels conservatively.

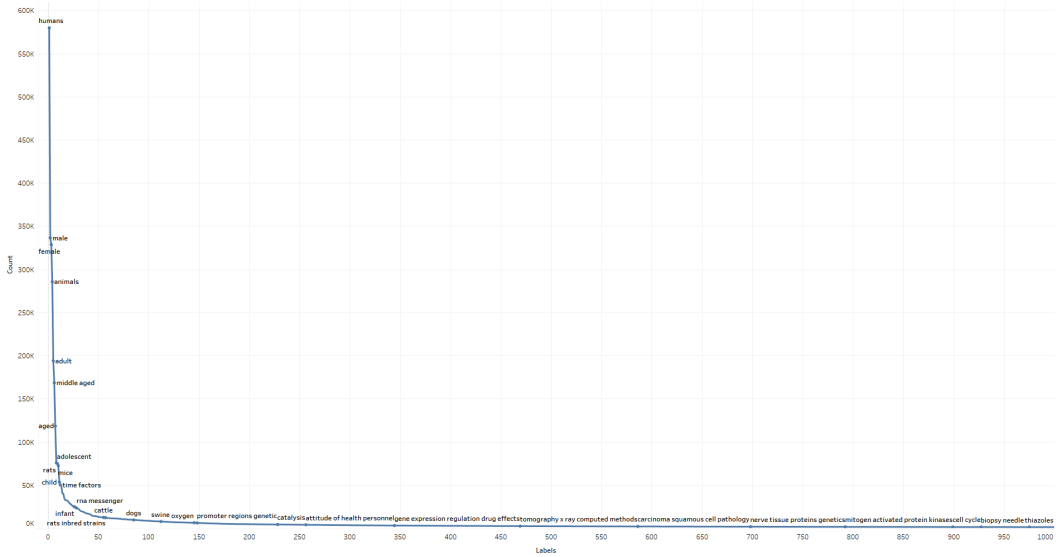


Fig. 3. Distribution of top 1000 most frequent labels in the subset of Medline

3.2.2 Label Density. Each document in the dataset is annotated with MeSH terms and can have anywhere between one and twenty terms assigned to it. On analysing the dataset, it was also discovered that there were records that had no subjects assigned to it. These records were expunged from the dataset prior to starting the experiments. For the remaining records, on average, the label density for the medline subset was seventeen labels per document.

3.2.3 Dataset Format. The dataset was made available as a tab-separated text file by OCLC B.V., where each line in the file corresponded to a record, with each attribute encased in brackets and preceded by a relevant keyword. For example, the title of the article in figure 1 would be encoded as *[title:The effects of eye and limb movement...]* in the text file. Other attribute keywords include *abstract, lang, author, pdate, issn, doi, citation, type, pmid,* etc. The keyword *subject* was used to refer to the MeSH headings in the dataset. The dataset was first read from the text file and the input text and labels were separated into different files and linked via unique IDs. The abstract and other meta-data were concatenated to form a single long paragraph of text. The concatenated word embeddings of this text would serve as the input to the model. The full medline dataset is freely available online for download via NLM, and can be obtained via bulk download, API, or as a small sample [37].

4 METHODOLOGY

4.1 XML-CNN

XML-CNN[32] is one of the first deep learning architectures to be tailored specifically for extreme multi-label classification. The proposed method for leveraging label relationships builds on the XML-CNN deep learning architecture.

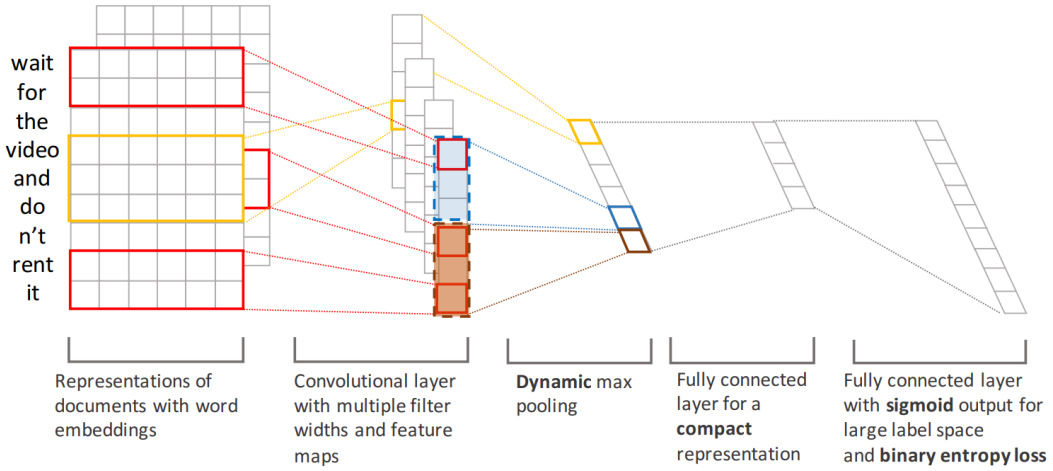


Fig. 4. XML-CNN Architecture as it appears in the original paper[32]

4.1.1 Model Architecture. The XML-CNN model consists of convolutional layers with multiple filter widths (two, four, and eight) that convolve over the document embedding matrix and generate feature maps that hold compact representations of the document, automatically extracting relevant features from input. The convolution filter sizes determine how many adjacent words are convolved over at a time, essentially functioning as an n-gram feature extractor. Max pooling layers follow the convolution layers and extract the most significant features from the feature map and feed into a hidden "bottleneck" layer, a hidden layer with the number of nodes far less than the max pooling and the output layer. This enables the document representation to be much more compact, avoiding a large set of weights (for a larger hidden layer) that would have otherwise slowed down computation and increased model size. The hidden bottleneck layer is connected to the output layer, which is the size of the label-space L and has a sigmoidal activation function. As in the original implementation, dropout was applied to the convolutional layers and the hidden layers in order to decrease the complexity of the learned model and prevent overfitting. The architecture of XML-CNN is shown in figure 4. The input for the model is the concatenated word embeddings of each document in sequence, constructed using a pre-trained Ariadne embedding, a Weighted Random Projections approach for Semantic Embedding[27].

The loss function used in the HXML-CNN model is the Binary Cross-Entropy(BCE) loss. The objective of BCE is to minimise:

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L}[y_{ij}log(\hat{y}_{ij}) + (1-y_{ij})log(1-\hat{y}_{ij})] \tag{1}$$

where n is the number of training samples, L is the total number of labels, y_{ij} is the target label j at instance i, and $\hat{y}_{ij}$ is the sigmoidal output of the label j at instance i. The binary cross entropy

loss calculates the error for each label independently and averages the losses for all labels to get an overall loss for the model. In essence, BCE treats each label as it's own binary classification problem (i.e, "is this label relevant or not?").

In multi-class classification, a common way of encoding the true labels for each training sample is to represent the labels as one-hot vectors of size L where each bit represents a unique label in the label-space and is set to '1' when it is present and all other bits are set to '0'. In multi-label classification, this encoding is extended to represent all true label bits as '1', creating a 'multli-hot' encoding representation of true labels. The same is true of extreme classification and the output vectors for XML-CNN are also multi-hot vectors with the size of the label space, L.

The source code for XML-CNN, implemented in Theano[52], is available on the extreme classification repository [53], which contains the details of the various parameters used in the original paper. The hyper parameters of the model such as the number of convolution filters, filter sizes, dropout values, embeddings sizes, number of hidden units, etc., were set according the values specified in [32]. Certain parameters were modified, such as the embedding dimension to enable better comparison to the performance of the model to the models being researched at OCLC and maximum input length to better fit the Medline dataset characteristics. The modified parameters were used for all models based on XML-CNN to ensure fair comparisons.

Training was conducted for 50 epochs, with an early stopping criterion applied such that the training stopped when the validation loss does not decrease by at least 0.0001 over 5 consecutive epochs. This ensured that if the model is slow to learn, there is an upper bound of 50 epochs for the model to train sufficiently while the early stopping made sure that the model does not over-train in the event of a plateaued loss. Additionally, a model checkpoint was also implemented that saved the best model (in terms of minimal validation loss) between epochs. These steps ensure that the best performing model on the validation loss is used for prediction and evaluation.

4.1.2 Input Embeddings. The original XML-CNN experiment by [32] used the GloVe[39] word embedding representations of raw text with little to no pre-processing techniques such as lemmatization or stemming or multi-word grouping being performed. The same approach is followed in these experiments. The only form of pre-processing done is to tokenize the input text using the Keras tokenizer in order to convert each token into a word vector. The Keras tokenizer performs a few basic text cleaning operations under the hood, namely, lower-casing of letters and filtering of uncommon symbols as well as some common characters such as punctuation. The resulting tokens were then converted to word-vectors using pre-trained word embeddings and concatenated to represent each input document.

The pre-trained embeddings used in the experiments is the Ariadne embedding[27], a Weighted Random Projections approach for Semantic Embedding, developed by OCLC. This embedding method uses a weighted random projection approach to generate embeddings. It was trained on medline to generate word vectors for domain specific terms. This is of importance as the medline dataset contains scientific jargon and highly domain specific terms varying across a wide range of medical fields of study. A lot of terms in medline are rarely used in day-to-day language, making their prediction a challenging task.

The embeddings used were of the dimension $D \times T$ where D is the embedding dimension and T is the size of the input text. The input text size is cut-off at a fixed length of 2084, the power of two closest to the average input text length of the dataset (avg. length was found to be 1858). Apart from a limit on the maximum input text length, a limit on the maximum vocabulary size to be used was set at 50k, similar to the original implementation. The embedding dimension used in the original XML-CNN implementation was 300, but the dimension of the Ariadne embedding was set to 256, as the embeddings being used at OCLC were of 256 and 512 dimension sizes. The

embeddings of size 256 was chosen as it was closest to the original. To enable fair comparison, the XML-CNN model was also trained with embeddings of size 256.

4.2 Hierarchical Approach

The XML-CNN approach ignores potential relationships between the labels themselves and trains to predict labels based solely on their independent relevance to each record. This assumption of label independence is violated in real-world applications, which often exhibit a meaningful relationship between labels. One way to leverage the label relationships is to use hierarchical trees that capture semantic relationships between the labels[29][6]. Such label-tree structures can be used to augment neural networks architecture by converting the linear output layer of the label-space to a hierarchical tree structure that groups labels into semantic clusters, with each leaf node representing a label. Hierarchical approaches have been used in language modelling tasks[34][33], multi-label classification tasks[29][6][28], and extreme classification tasks[38][41] to great effect. Note that there two kinds of hierarchical approaches:(1) Methods discussed in section 2.2.3 that learn classifiers to partition the training instances directly, and (2) Methods that extract hierarchies by partitioning the label-space in order to extract inter-label relationships. This project mainly focuses on the label-space partitioning approach. Both the existing MeSH subject heading hierarchy, as well as automatically generated hierarchy trees are considered in the experiments. Each node in the label hierarchy is characterised as belonging to a "layer" (a horizontal collection of nodes) over which the loss function is applied, with each layer calculating it's own loss during training. The intuition behind grouping the nodes in such a way is that the hierarchies can be expected to have more general (or more frequent) terms near the root of the hierarchy and the more specific (or less frequent) terms near the leaves of the hierarchy. The grouping of the nodes into layers attempts to leverage this relationship between the nodes such that the more general nodes will inform a better hidden layer representation for the specific nodes through Mutli-Task Learning (MTL), considering the prediction of more general and frequent labels to be a separate task from the prediction of the more specific and less frequent labels.

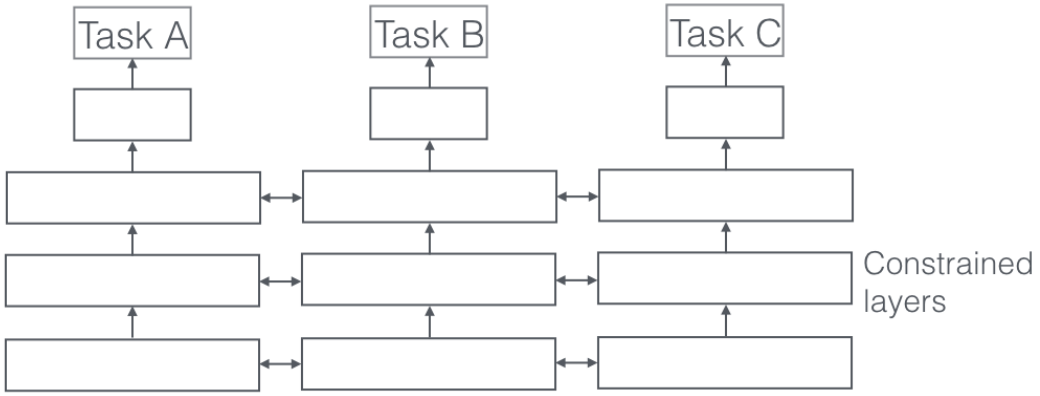


Fig. 5. Example architecture illustrating Soft Parameter Sharing [45]

4.2.1 Multi-Task Learning. Multi-Task Learning is a subset of machine learning where, instead of focusing on one single task or metric, the same model is used to learn multiple tasks, usually with some shared parameters and a separate loss function per task. The sharing of parameters between the different tasks forces the neural network to learn generalised representations for all

tasks, thereby reducing the risk that the network overfits on one of the tasks. MTL has a large list of applications from NLP, speech recognition, computer vision, etc[16][17][20]. MTL for deep learning methods employs one of two methods: *Soft Parameter Sharing* and *Hard Parameter Sharing*. Soft parameter sharing involves using unique models with their own parameters for each task, where the distance between the parameters of models is regularised to encourage similarity (fig. 5). Hard parameter sharing on the other hand, involves sharing the hidden layers amongst all tasks and having task-specific output layers (fig. 6). This approach is the most commonly used MTL method and is adept at reducing the risk of overfitting to a particular task [7]. As hard parameter sharing results in a smaller model size and number of models compared to soft parameter sharing, hard parameter sharing is used in the experiments.

The application of MTL to MeSH subject prediction can be achieved by considering the prediction of the different levels of general/specific labels in a label hierarchy to be slightly different, but related tasks. The use of label hierarchies inform the differentiation of the prediction tasks based on the label specificity. Through hard parameter sharing, the training of the model on predicting the more general labels (Task A) is expected to bolster the shared hidden layer representation and enable better prediction of the more specific labels (Task B) that are harder to predict on their own.

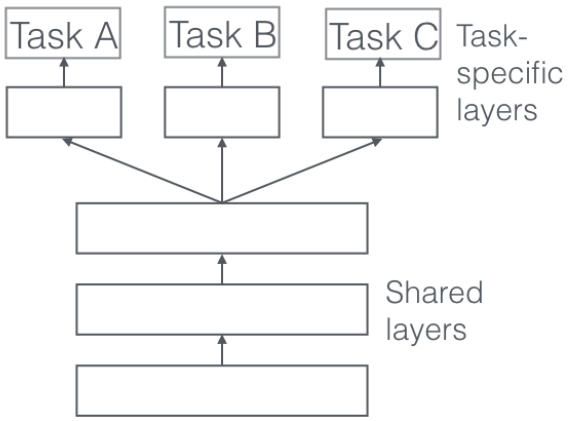


Fig. 6. Example architecture illustrating Hard Parameter Sharing [45]

4.2.2 Automatically Generated Hierarchy. Most large-scale datasets do not have well-maintained label hierarchies and therefore, must rely on automatically generated hierarchies. Automatic hierarchies can either be generated by using existing linguistic resources such as WordNet[34] and Wikipedia[49] (for datasets with no existing labels) or through polythetic, hierarchical clustering algorithms on the label space (for datasets with labels). For this project, the latter approaches are considered. These approaches use different algorithms to generate hierarchies, such as a balanced k-means clustering[38][40], hierarchical agglomerative clustering[30], etc.

The automatic label hierarchy for this project was generated by applying *Hierarchical Agglomerative Clustering* (HAC) on the Ariadne embeddings of the labels. Since the Ariadne method embeds labels based on a global co-occurrence matrix, it is reasonable to expect that similar labels can be expected to be closer to each other in the embedding space. Since HAC groups labels in a bottom-up approach by clustering items that are close to each other, similar labels are more likely to be forced into the same or nearby clusters. While HAC is known to have large computational complexity that makes it slower than k-means clustering, HAC is chosen as it readily provides a binary tree

structure dendrogram, does not need the number of clusters to be pre-defined, is repeatable and consistent, and isn't constrained by the need for spherical data as k-means is.

The results of HAC can vary based on the linkage methods used to combine clusters, such as *single linkage*, *complete linkage*, *average linkage*, and *Ward's method*. Of these methods, *Single linkage* is best avoided for large label-spaces as it is prone to chaining of nearby labels in the embedding space, forming loose clusters, *complete linkage* produces tight clusters, and Ward's method is incompatible with non-Euclidean distances[30], such as the Cosine distance that Ariadne uses. Therefore, average linkage is chosen. Average linkage works by calculating the average distance between each point in a cluster and all other points in all other clusters, joining the two clusters where the average of these distances is the lowest.

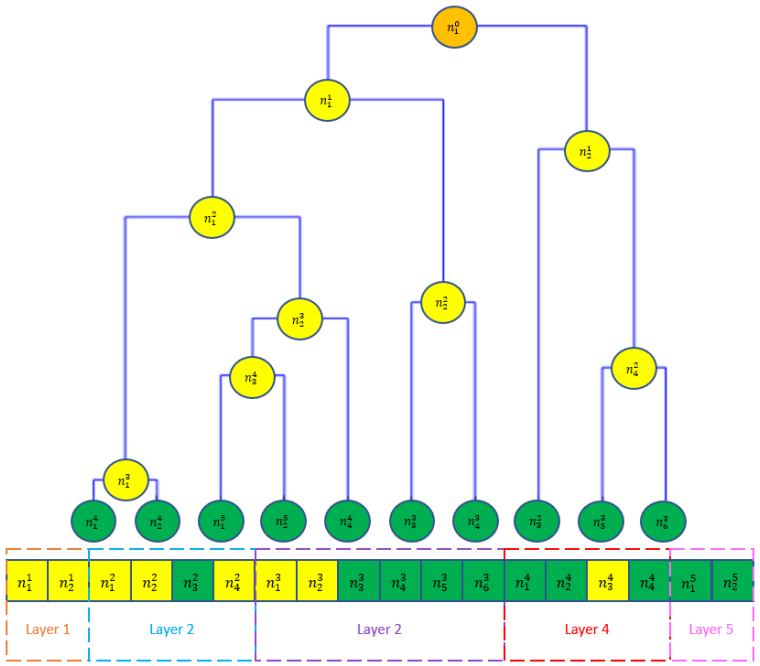


Fig. 7. Sample dendrogram for 10 labels with top-down layer grouping. Green nodes are leaf nodes/labels, and yellow nodes are internal nodes/clusters. The superscript in the node name indicates the layer to which the node belongs to, and the subscript indicates the node-id when counting the nodes left to right within that layer.

HAC results in a *dendrogram* (binary tree) that groups all labels at the leaf nodes with each cluster as an internal node. A sample dendrogram for 10 labels is shown in figure 7 with the green nodes representing the labels and the yellow nodes representing the clusters. There are two main ways the nodes are grouped:

(1) *Top-Down:* In this method of grouping, the nodes that have the same depth (number of edges in the longest path to the root) are considered to be part of the same layer. This results in a grouping that has leaf nodes spread out over the layers, depending on the number of ancestor nodes of each node. The layers are counted from top to bottom, so the first layer will contain the immediate child nodes of the root, and so on. The intuition behind this method is that

the clustering will separate the general terms from the specific ones and this separation will be captured in the layers.

(2) *Bottom-Up:* In this method of grouping, the nodes that have the same height (number of edges in the longest path to any leaf node) are considered to be part of the same layer. The result of this method is that all the leaf nodes are considered to be part of the same layer. The layers in this case are counted from the bottom to top. In this case, the relevant labels are grouped together in one layer while the internal nodes form layers based on their position in the hierarchy. The idea is that since alpha will be penalising layers, the relevant labels will need to have the least penalty.

The naming convention for each node is n_b^a, to represent the b^{th} node of layer a, counted from left to right. For example, in the top-down grouping method, node n_2^3 in fig.7 is the second node from left to right in the third layer as it has three parent nodes above it, including the root node. These layers are then used to construct the output layer of the modified XML-CNN architecture. When using HAC on the training label-space, the total number of output nodes of the XML-CNN model is doubled due to the addition of the internal nodes of the hierarchy.

4.2.3 MeSH Hierarchy. The relationship between MeSH subject headings can be represented as a hierarchical tree structure[35]. The tree has sixteen main heading branches including Anatomy, Chemicals and Drugs, Diseases, Humanities, Organisms, etc. These main branches represent a broad umbrella under which further specific headings are placed. For example, consider the term "Torso". This term is placed beneath the main term Anatomy, under the heading "Body Regions". Traversing further down this branch, we find the term "Back", under which are more specific terms such as "Lumbosacral Region" and "Sacroccoccygeal Region". The leaf nodes of the tree contain highly specific terms that reveal more interesting descriptions of an article. All headings in the MeSH tree structure are not necessarily part of a unique branch. There are many headings that appear on more than one branch of the tree. For example, the heading "Eye" appears in both the "Body Regions" branch as well as the "Sense Organs" branch.

The hierarchy is treated in the same way as the automatically hierarchy where each node is grouped into layers depending on its depth or height. The difference in this case is that while the internal nodes in the HAC tree were not actual labels but rather clusters (that could be said to represent an abstract concept representative of all inhabitants of that cluster), the internal nodes of the MeSH tree are actual labels that have semantic meaning and can be assigned to a record if so desired.

The MeSH subject heading hierarchy tree is maintained manually and updated along with the MeSH terms by NLM and consists of all the MeSH headings used by NLM indexers. As such, the hierarchy contains labels that are not present in the label space, and increases the total number of output nodes from twenty-two thousand to nearly thirty-eight thousand. The additional fifteen-thousand new labels do not exist in the training label space of the medline sample dataset used, but can be found in other records in the full medline dataset. These nodes were nevertheless included in the tree structure as their removal meant that the tree structure, and therefore, the layer structure would be altered. These extra labels were excluded from the final evaluation in spite of the possibility that they may be semantically relevant to the record. This is done because these labels never appear in the partial ground truth of annotated labels, and hence, will have a negative impact on the evaluation metrics used even if they are semantically relevant.

Another difference in the MeSH hierarchy is that some labels in the hierarchy do not have a unique position, i.e., the same label may hold multiple positions in the hierarchy due to their

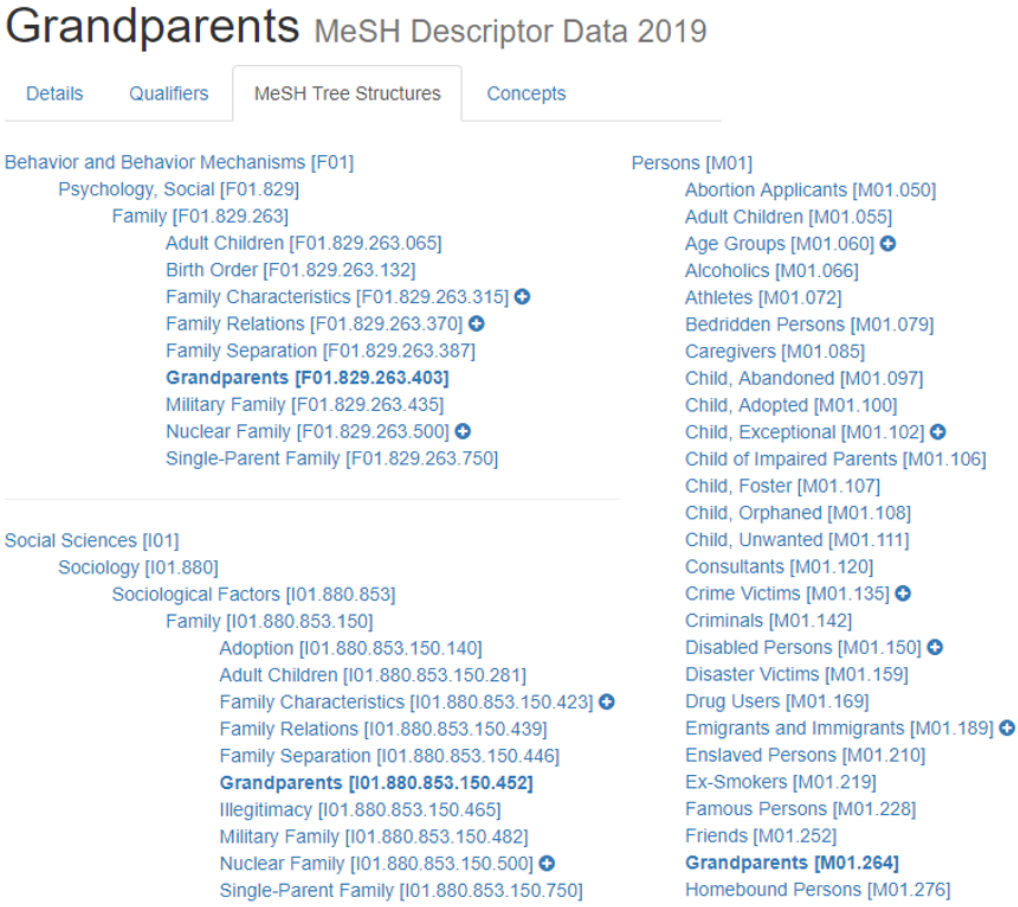


Fig. 8. Duplicate nodes at varying depths in the MeSH Hierarchy for the MeSH heading 'grandparents'. Image taken from the MeSH Browser[35]

meaning in different contexts. For example, the term 'grandparents' can be seen in the hierarchy under three different root nodes: 'Behaviour and Behaviour Mechanisms', 'Social Sciences', and 'Persons' (fig. 8). Each of these have a different context that changes the meaning of the label in regards to it's usage as a descriptive tag. However, a key problem is that the correct position of the label as it is used for each record is not easily identifiable, as the NLM indexers only tend to use the most specific labels for annotation (thereby ruling out the possibility of identifying if a label co-occurs with it's ancestors). Due to this issue, these duplicates were treated as effectively the same term regardless of context, i.e., if a label is annotated and has multiple nodes associated with it, all nodes are given equal consideration.

4.3 Hierarchical XML-CNN

In order to use the label hierarchy in the XML-CNN architecture, the hierarchy is flattened and concatenated layer-wise and used as the replacement for the output layer of XML-CNN in figure 4. The hidden bottleneck layer is densely connected to this structure, as shown in figure 10.

As the output layer now includes additional nodes compared to the actual label-space, these nodes will never be 'active' for any record as they are not part of the annotated labels list. To provide a reasonable way to bring these nodes into play during training, the multi-hot encoding of the target labels is modified slightly. When generating the multi-hot representations of the labels for training, validation, and testing, an internal node is set to 1 if, for the current training instance, at least one of its child nodes is present in the list of annotated labels. This yields different effects for the automated and the MeSH hierarchies:

(1) *HAC:* In the automatically generated hierarchy, since all internal nodes aren't actual labels, this approach of artificially inflating the count of ancestor nodes doesn't affect the actual dataset since no label from the list of annotated labels will have an inflated count. Therefore, the approach gives a reasonable method for bringing the ancestor nodes into the training process.

(2) *MeSH:* In the MeSH hierarchy, this method doesn't quite have the same effect due to the fact that internal nodes are also actual labels that have meaning, and may be in the list of annotated labels for the dataset. This results in the issue that labels that are not leaf nodes will have an inflated count overall, and therefore, more samples to work with, while leaf nodes will have the same frequency as before.

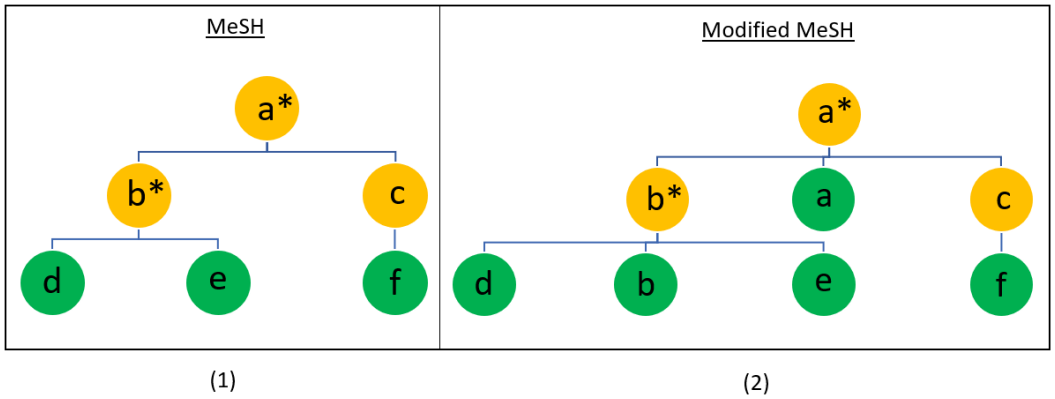


Fig. 9. Handling internal nodes (yellow) that are part of the label space in the MeSH hierarchy. (1) depicts the scenario where internal nodes a* and b* are part of the label space and (2) shows the workaround scenario where proxy leaf nodes *a* and *b* (green) are added as child nodes to a* and b*.

This inflation in the annotation frequencies gives certain nodes an unfair advantage, which makes it difficult to compare the performance of the MeSH HXML-CNN model with the baseline and with the HAC HXML-CNN model. In order to enable fair comparisons between the MeSH and HAC hierarchies, it is necessary to ensure that the artificial 'boost' to the frequencies of the internal nodes in MeSH is circumvented. In order to achieve this, internal nodes that are part of the training label-space are modified by creating an artificial child node with the same name, which functions as a proxy-leaf-node for the internal node. The proxy-leaf-node is then treated as the annotated label and is used in evaluation instead of the internal node. This way, the boost in frequencies only affects the internal nodes, and does not influence the evaluation of the predicted

proxy-leaf-node. This workaround is illustrated in figure 9. The use of the MeSH hierarchy with the modified internal nodes leads to an increase in the output nodes of the model to almost four times that of the non-hierarchical output layer of XML-CNN. In addition to the fifteen-thousand MeSH headings that were added to the original label space, this large increase is due to the fact that the MeSH hierarchy has non-unique nodes that have been duplicated to avoid count inflation. The total number of output nodes in the MeSH hierarchy was approximately seventy-seven thousand.

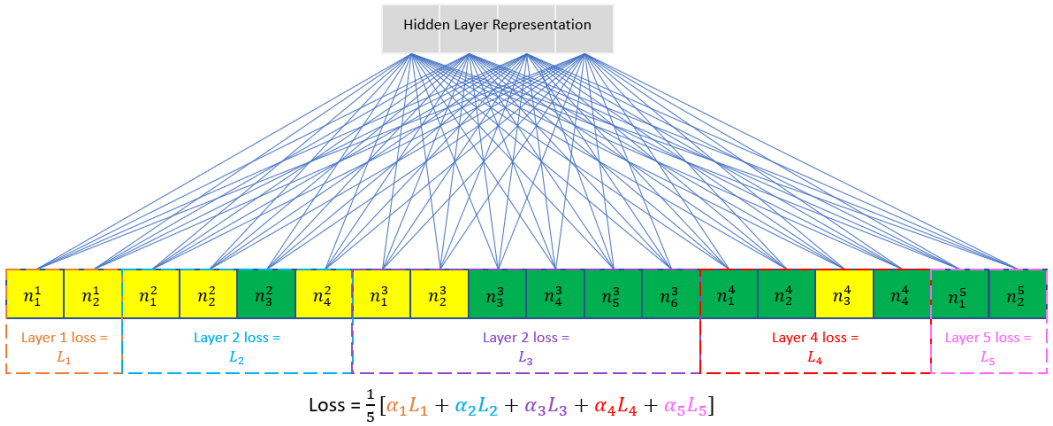


Fig. 10. Flattened label hierarchy from figure 7 as the densely-connected output layer (for 5 layers). Each loss function ($L_1 ... L_5$) is the binary cross-entropy over sigmoid activation for the corresponding layer. $\alpha_1 ... \alpha_5$ are hyper-parameters for each layer.

4.3.1 Alpha. The loss function used is the binary cross-entropy loss over sigmoid for each layer, as shown in figure 10, where the total loss of the model is the mean of the losses of each layer. The loss for each layer is the Binary Cross Entropy loss function, given by equation (1). The overall loss, shown in figure 10, introduces α_1, $\alpha_2 ... \alpha_a$ as a penalty factor. The intuition behind alpha is that certain output nodes in the label space are of more importance than others, namely:

(1) The nodes that represent infrequent labels are more desirable than those that are highly frequent. Hence, alpha should penalise frequent labels more than infrequent labels.

(2) The nodes that represent the training label space (relevant labels) are more important than the labels that represent labels absent from the training label space. Hence, alpha should penalise the non-relevant internal nodes more than relevant nodes.

The value of alpha varies per layer to reflect these differences and hence, was calculated as a function of the frequency of the labels in a layer as well as the number of relevant labels in that layer (where relevant labels mean all labels that are part of the set of all training labels in the dataset). Alpha is inversely proportional to the frequency of labels and directly proportional to the number of relevant labels per layer.

$$\text{If, for each layer in } L_0, L_1, L_2, ..., L_a :$$
$$c_a = \textit{Number of relevant labels in layer 'a',}$$

f_a = *Sum of frequencies of all relevant labels in layer 'a',* and

k = *Constant to fine-tune the value of alpha,*

$$\text{Then, for layer '}a\text{':}\ \ \alpha_a = k * \frac{c_a}{f_a} \tag{2}$$

Once the raw alpha values are obtained, they are then normalised using min-max normalisation in order to enforce a range of $[0, 1]$. The constant k is set to 1 and can be varied to get larger or smaller values for all α. In addition to this method of calculating alpha, the following were also tried:

(1) *Alpha = 1 for all layers*: This essentially means that there is no explicit penalty applied and this value of alpha would treat the HXML-CNN model as the same as XML-CNN with additional output nodes. This configuration could show the effect of MTL when using label hierarchies.

(2) *Alpha based on a steady decay over layers*: In this approach, alpha is decayed as 0.9^x, where $x = 0, 1, 2, ..a$ for layer a and $x = 0$ for the layer containing the most leaf nodes.

(3) *Alpha based solely on the frequencies*: Here, alpha is calculated the same as before, but this time, c_a is set to 1 so that the influence of relevant labels becomes irrelevant. This is done because the bottom-up configuration renders the above calculation of alpha invalid due to the concentration of all relevant labels in a single layer. The frequency based penalty implicitly also penalises the layers with more internal layers due to their inflated frequencies.

The original calculation for alpha as shown in eqn.(2) was eventually discarded as the values of alpha for layers was too extreme and resulted in poor performance. This is discussed in detail in section 4.7.3

4.4 Environment Setup

The environment setup consists of two distinct set of procedures: (1) installing Python and the relevant python libraries for the experiments, and (2) installing other supplementary software. All libraries and software installed were the latest versions at the time, with the exception of Python and the Nvidia Cuda and cuDNN packages for GPU usage, which needed to be installed according to the compatible versions specified by TensorFlow on their website[1]. The full list of libraries and software used, along with their relevant web-urls is given in table 2, in appendix A.1.

4.4.1 Python Libraries. The following python libraries needed to be installed for the environment setup:

(1) Tensorflow: The experiments were to be written in Python using TensorFlow[1], a widely used open-source library for machine learning, developed and supported by Google. TensorFlow offers a high-level API, Keras[14], through its *tf.keras* module, which makes building deep learning models extremely modular. Keras is much more intuitive than other TensorFlow high-level APIs and can be used to quickly build deep learning model prototypes. TensorFlow provides two types of installation packages, one for use with CPUs and another for use with GPUs. The GPU version is used for this project for faster training and evaluation. The GPU version requires the following Nvidia packages to be installed before TensorFlow - Nvidia CUDA Toolkit and cuDNN SDK. The respective versions for these packages is documented on the tensorflow website. Additionally, the Nvidia drivers of the GPU need to be up-to-date before the installation begins. Issues with the installation of tensorflow-gpu are commonly

due to version mismatches and outdated installations of Nvidia components and drivers.

(2) Standard ML Libraries: In addition to Keras for Deep Learning, other standard python libraries commonly used for machine learning pipelines used in this project include *NumPy* for scientific computation, *Pandas* for high-dimensional data manipulation, *Scikit-Learn* for evaluation metrics and other ML tasks, *Gensim* for word embedding manipulation, and *Seaborn* for plots.

(3) Special Libraries: Apart from the traditional python libraries mentioned above, a few special libraries were also used in order to help with certain aspects of the experiments. These include the following - *h5py* for saving and loading keras models as HDF5 files and *iterative-stratification*[4] in order to implement stratification of the label space in train-validation-test splits. The latter is discussed in further detail in Section 4.5.2.

4.4.2 Additional Software. The following applications were installed to ensure a smoothly running working environment.

(1) Package Manager: In order to manage the different versions and dependencies of packages when installing multiple different libraries, having a package manager can save time. For this project, Anaconda, an open-source Python distribution was used, which uses the conda package manager to keep track of dependencies and versions of an environment. Anaconda also allows users to create different environments for different purposes, such as for legacy python versions.

(2) IDE: The development environment for the project included two IDEs - Jupyter Notebook and PyCharm Student Edition. Jupyter notebooks were used for the initial prototyping and later as a playground for minor, one-time tasks that required quick testing. PyCharm was used as the main IDE for development and running experiments. While the Jupyter Notebook was useful for small experiments and minor tasks, PyCharm was necessary for a structured and systematic approach to running multiple experiments and maintaining a structured code-base.

(3) Version Control: Git was used as the version control of the code-base for the experiments. A GUI application for git, SourceTree was used for easier usage. The version control was only used for the actual code and small supplementary files, and not for large files such as the datasets and train-validation-test splits, as these files remained unaltered once created.

4.5 Handling Large Datasets and Label Spaces

4.5.1 Data Generators. While smaller datasets could be trained easily in Keras using the *model.fit* method by loading the dataset into memory, the same is impossible to do for medline or other extreme-multi-label datasets. The Medline dataset is a substantially large dataset; even the 1 million subset of medline is large enough that it cannot be stored in memory for training and evaluation. However, due to the large label space, the size of the output vector is also extremely large, adding to the already large size of the dataset.

In order to get around the memory problem, the dataset had to be read from the disk in batches during batch-training using a python generator and trained using the keras *model.fit_generator* method. This approach, while resolving the problem of fitting data into memory, was extremely slow to train, as reading the data from the disk during GPU training meant that the GPU was idle for long times between epochs when the system waited on the CPU for creating the next batch of

data for training. However, a better, more structured approach for dealing with large datasets is described in [3], which extended the keras *Sequence* object for creating data generators and made use of multiprocessing. The keras *Sequence* object is a safer way to utilise multi-processing that guarantees that the model will only train once on each sample per epoch (unlike regular python generators). This structuring was used for the experiments for parallelized data loading during training, significantly decreasing training time.

4.5.2 Stratification. When splitting the dataset into train-validation-test sets, random splits gave rise to the problem of insufficient training samples and/or missing labels in the splits; due to the label sparsity, some labels that were present in one of the sets (train/validation/test) would be missing in one or both of the other sets. To get around this problem, the stratification of labels was performed as described in [46]. A python library for multi-label stratification is available at [4]. Stratification tries to ensure that the label ratios in all three splits would be maintained, i.e., the algorithm tries to preserve the distribution on labels between the splits. Of course, for some of the extreme tail labels (such as those with under 20 samples), there would still be issues of missing labels, such as the test set containing no positive examples (rendering precision and recall for the label useless). This is because the library only allows for splitting a dataset into two using stratification. Therefore, in order to get three splits (train, validation, and test sets), stratification had to be applied in succession, which lessens it's effect somewhat. Additionally, this method was designed primarily for multi-label datasets and it's effects are slightly less reliable in the extreme setting due to the large label space. Nevertheless, the use of stratification is justified as it gives the model a better chance of learning and evaluating the input to label mappings [46] and was found to be better than random splitting in prior experiments. Using stratification, the dataset was divided into train-validation-test splits.

4.6 Evaluation

Traditionally, classification problems use evaluation measures such as accuracy. However, accuracy would not be useful in the XMTC scenario due to the heavily imbalanced class structure arising from the power law distribution of labels. In such cases, accuracy will favour the majority class and the model can achieve a reasonably high accuracy by just predicting the majority classes of the distribution and ignoring the minority class[13]. This rules out the use of such metrics.

In XMTC problems, though the label-space is extremely large, each record will only have a handful of labels. As mentioned in section 3, the average label density of the dataset was seventeen. With such a short set of labels (compared to the label space) to be predicted per article, and with the goal of selecting labels that are most descriptive of the record, it is important for the selected labels to be sorted in the decreasing order of their predicted confidence (represented by the sigmoidal output at each corresponding output node) so as to select the most likely and relevant labels out of the large label space, i.e., a ranked list of predictions is desirable. This ensures that in practical scenarios, a reviewer/indexer can quickly go through the list of predicted labels and evaluate the predictions by only looking at the top part of the ranked list. Ranking metrics such as Precision@K and Recall@K are more appropriate for XMTC, and have been used widely in literature for such problems[32][42][5][57][58][50]. The most commonly used value of K for ranked recall and precision are Precision@1, Precision@3, Precision@5, Recall@5, Recall@10, and Recall@20.

For the final evaluation, the predicted labels were sorted by their predicted confidence, the top 100 were selected and the *Average Precision@K* and *Average Recall@K* were calculated for all selected labels as the average of the recall for values of *k=1, 2,...100*. In the cases where the labels had no positive examples in the test set due to the train-validation-test split, the label was ignored.

In addition to selecting the top 100 predictions, for the predictions made by the MeSH HXML-CNN models, certain other rules are used to filter the predicted labels before evaluation. Since the predictions made by MeSH HXML-CNN include labels that are not part of the label space, certain rule based filters are required to ensure fair comparison with the other models. The following rules are applied to these labels:

(1) Only the labels that are part of the training label space (relevant labels) are selected for evaluation. Other labels are discarded.

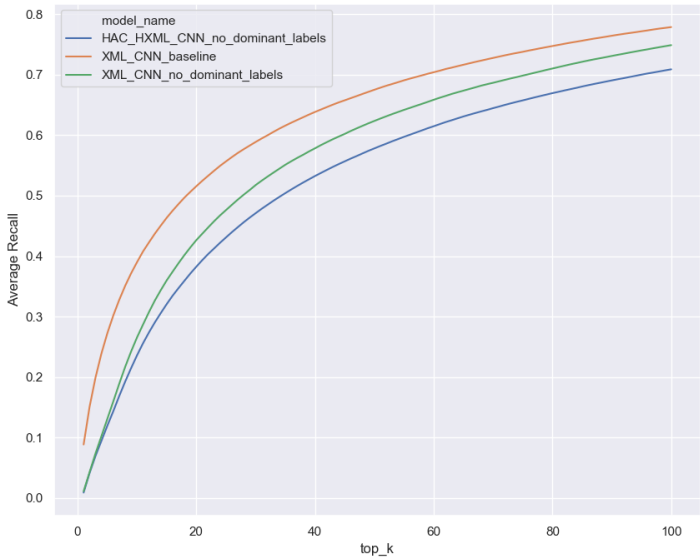
(2) If a both a child node and one or more of it's ancestor labels (proxy-leaf-nodes) are predicted, the child node is selected over the parent as child labels are expected to be more specific in nature and hence, more descriptive of each record.

(3) As an exception to Rule #2, if any of the predicted ancestor labels are check-tags or head labels (freq > 5% of total training records), they were included in the final list regardless of whether a child node was also predicted. The reasoning for this is that these labels make up a large portion of the ground truth, and not predicting these labels can have a negative impact on the performance metrics.

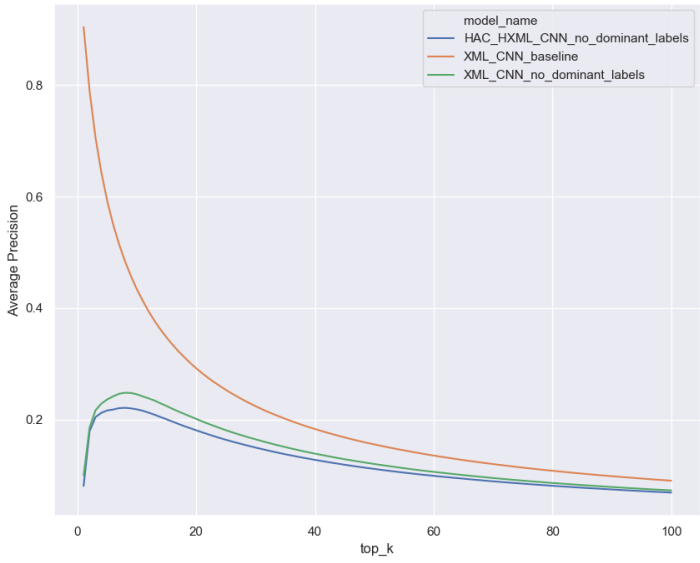
4.7 Experiments

The Keras implementation of XML-CNN, modelled after the Theano source code of XML-CNN from the extreme classification repository[53], was run on medline and used as a baseline to compare the performance of the hierarchical xml-cnn. The performance of the vanilla xml-cnn model on medline in Precision@K (K=1,2,3) and Recall@K (K=5,10,20) was calculated for comparisons. The hierarchical models were then constructed with both the HAC dendrogram and the MeSH tree. The hyper-parameters for each model were set according to the specifications of the original XML-CNN model in order to enable fair comparisons. The initial round of experiments included certain configurations that were ultimately rejected due to poor performance. These are detailed below:

4.7.1 Excluding Frequency Dominant Labels. One of the initial experiments was to separate the tasks of predicting the high frequency labels and the moderate-low frequency labels separately. The prediction of the high frequency labels wasn't a concern as there were enough training examples for these labels to be predicted with high precision and recall. Prior experiments had shown that these labels often dominated the predictions. One set of experiments involved removing high frequency labels (with frequency > 5% of the total training records) from the label space and focusing on the rest of the labels. Overall, by this method, the top twelve most frequent labels were removed from the label space. For this label space, the experiments were tested for XML-CNN and HAC-HXML-CNN models. The results are shown in fig.11a and fig.11b. The recall plot shows that both models experienced a dip in performance when the most frequent labels were taken out, which was expected - highly frequent labels make up a significant portion of the annotated labels, and without them, performance was expected to suffer. The plot for precision is more interesting, and reflects the need for Rule#3 in section 4.6. The precision of both XML-CNN and HAC-HXML-CNN started out incredibly low, indicating that the first few labels, i.e., those predicted with the highest confidence for both models, tend to be the head labels. This means that predicting these labels is essential to scoring high on the chosen performance metric.



(a) Average Recall@k



(b) Average Precision@k

Fig. 11. Average Recall@k and Average Precision@k (k = 1,2,3,..100) for XML-CNN and HAC-HXML-CNN models with no dominant labels. XML-CNN baseline evaluated on all labels is also shown for reference.

4.7.2 Top-Down Layer Configurations. The top-down configuration that would group together the labels that share the same depth was used for both hierarchies with the assumption that it would separate the general labels that would be closer to the root of the hierarchy from the more specific ones. However, both for the HAC and MeSH hierarchies, the top down configuration led to multiple labels being included in layers with high alpha penalty. The result was a heavy drop in performance due to labels being penalised heavily. The top down configuration was discarded in favour of the bottom-up configuration, that was much more resistant to the problem of high layer penalty.

4.7.3 Alpha Penalty. The calculation of alpha as described in equation (2) resulted in values of alpha that were too extreme. The performance for both the HAC and MeSH HXML-CNN models were impacted negatively as the high penalty meant that the model wasn't able to learn reliable mappings from the input records to the output labels. Additionally, since the top-down configuration was abandoned, the calculation of alpha based on the number of relevant labels in each layer ceased to be useful as the bottom-up configuration grouped all relevant labels into one layer. Instead, alpha was calculated based solely on frequency. Other methods for calculating alpha have been discussed in section 4.3.1.

Apart from these experiments, a few more experiments were run, such as testing the original MeSH hierarchy without the modification described by fig.9 in section 4.3. However, since the original MeSH hierarchy had the problem of artificial inflation of their training samples, these experiments were incomparable with other methods and hence, were discarded. The final experiments of significant interest all involve the bottom-up configuration for the HAC-HXML-CNN and MeSH-HXML-CNN models with the three alternative methods for calculating the alpha penalty described in section 4.3.1. The performances of these models is described in the next section.

Method Name	Alpha per layer	Average Precision			Average Recall		
		P@1	P@3	P@5	R@5	R@10	R@20
XML-CNN	-	0.90	0.71	0.60	**0.27**	**0.40**	0.52
Ariadne	-	0.90	**0.75**	**0.66**	0.26	**0.40**	**0.54**
HAC HXML-CNN	1	**0.91**	0.69	0.57	0.26	0.37	0.49
	0.9^x	**0.91**	0.71	0.59	**0.27**	0.39	0.51
	$\propto \frac{1}{\text{sum of freq.}}$	0.66	0.44	0.31	0.14	0.17	0.21
MeSH HXML-CNN	1	0.90	0.69	0.58	**0.27**	0.38	0.51
	0.9^x	0.89	0.69	0.58	**0.27**	0.38	0.50
	$\propto \frac{1}{\text{sum of freq.}}$	0.88	0.67	0.57	**0.27**	0.37	0.50

Table 1. Results

5 RESULTS

The results of significant experiments are discussed in this section. Each model tested is compared with the performance of the original XML-CNN model in Precision@k(k=1,3,5) and Recall@k(k=5,10,20). The overall results of significant models is given in Table 1. The Average Recall and Average Precision for k= 1,2,3,...100 are also plotted for all significant models in figures 12, 13, 14, and 15. These plots are primarily for visual reference, and not indicative of model superiority past k=20, as most practical applications will only need the top 20 or ranked predictions. In addition to the deep learning models, the performance of the Ariadne method for MeSH subject prediction provided by OCLC is also included in the results for comparisons. From the overall results, it can be seen that none of the HXML-CNN models were able to improve their performance over the original XML-CNN model. The best performing model and the most robust is the Ariadne method. The inclusion of the hierarchies, whether automatically generated or manually created, do not appear to improve the performance of the deep learning models over the original model. However, the performance of both hierarchical models appears to be respectably close to that of XML-CNN.

5.1 HAC Tree

While the HAC-XML-CNN model did not quite beat the XML-CNN model, its results were quite similar to that of the XML-CNN model. The best performance for the HAC-HXML-CNN models is with the bottom-up configuration and alpha that was decayed by 0.9^x per layer. The Average Precision and Recall at k=1 and k=5 (respectively) for this model is either better than or equal to the best score, but both metrics drop slightly as k increases. However, the high value for the P@1 and R@5 could be attributed to the prediction of the dominant head labels. Regardless, the performance of this model is nearly identical (1% difference) to the baseline XML-CNN model.

The next best HAC-HXML-CNN model has the bottom-up configuration with alpha = 1 for all layers. This model, while close to the baseline, consistently under-performs compared to both the baseline, and the HAC model with the steadily decayed alpha. Just as the previous model, the Precision@1 for this model is also one of the best. When the alpha penalty was calculated as shown in equation (2), the performance of the model fell drastically. Overall, the HAC-HXML-CNN model was able to keep up with the performance of the baseline XML-CNN model in-spite of the number of output nodes being doubled by the use of the hierarchy.

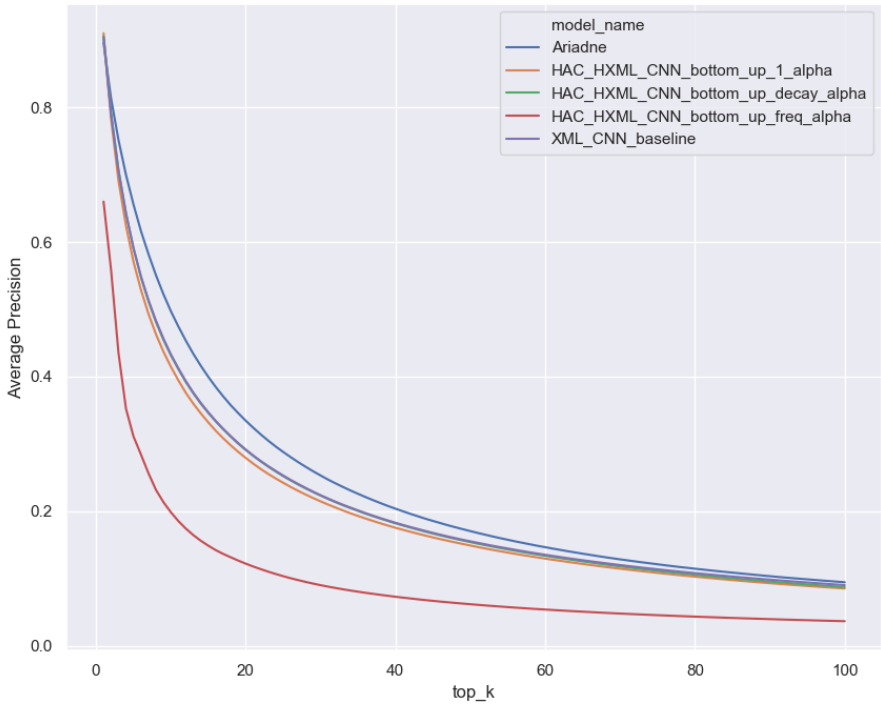


Fig. 12. Average Precision@K (for K=1, 2,...., 100) for the significant HAC models trained on the medline dataset

5.2 MeSH Tree

Similar to the HAC Tree, experiments run with the MeSH tree also yielded results that were close to that of XML-CNN, however, none of the MeSH-HXML-CNN models were able to quite beat XML-CNN. The best performing model for the MeSH hierarchy was the bottom-up configuration with alpha=1 for all layers. Even with the number of output nodes increasing nearly four times as much as that of XML-CNN, the performance did not suffer drastically. While this model was bested by the HAC-HXML-CNN model with steadily decayed alpha, it's performance is certainly respectable and close to the baseline.

The MeSH hierarchy also appeared to be more resistant to the penalty enforced by the alpha parameter, as all three calculations of alpha yielded similar performance in the bottom-up configuration. While the frequency based alpha penalty significantly reduced performance in the HAC-HXML-CNN model, it only had a slight negative effect on the MeSH-HXML-CNN model. This could be due to the presence of non-unique nodes in the MeSH hierarchy. As a significant portion of the output nodes in the MeSH hierarchy are duplicates that are inseparable, the hierarchy could be masking the effect of the alpha penalty on multiple internal nodes due to their differing positions in the hierarchy. With different nodes (that refer to the same label) being penalised differently, the effect of the penalty may have been diluted somewhat in the MeSH hierarchy.

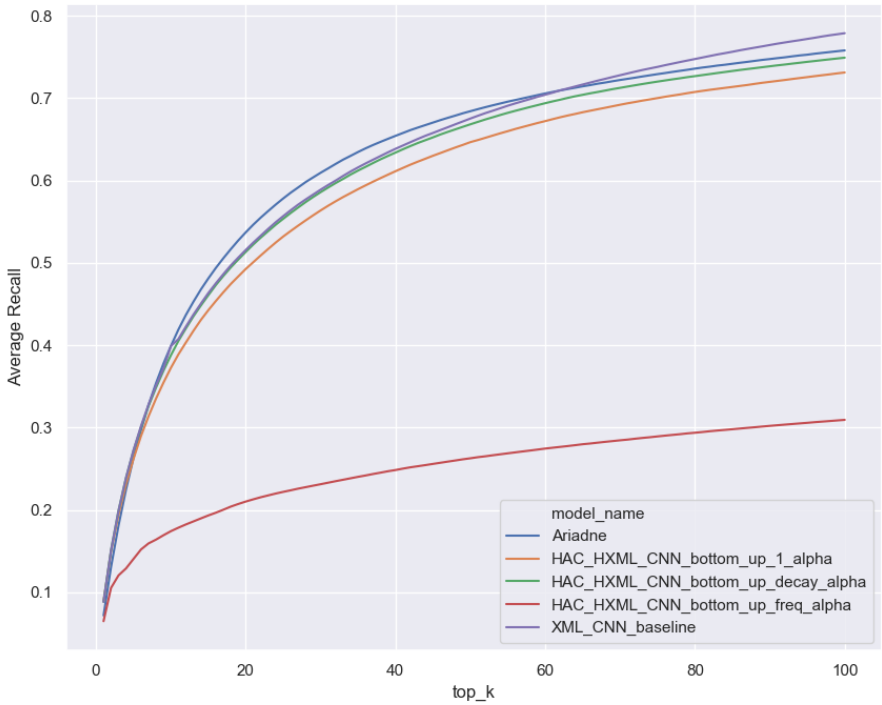


Fig. 13. Average Recall@K (for K=1, 2,...., 100) for the significant HAC models trained on the medline dataset

6 DISCUSSION

6.1 Hierarchical XML-CNN

Overall, the results of the experiments show that the usage of a hierarchy for the prediction of MeSH headings for Medline is close to the state-of-the-art deep learning model XML-CNN, but not better (**RQ 1**). The results of the automatically generated tree by Agglomerative Clustering is much closer to XML-CNN than models that use the MeSH hierarchy, but not by much(**RQ 2**). This is promising because the usage of other artificial hierarchies in the same method may outperform the XML-CNN model depending on the resulting clusters. Specifically, the HAC algorithm forces a binary tree. This binary restriction on clustering is not reflective of how the labels for Medline ought to be clustered, i.e., label hierarchies are rarely going to be perfectly balanced binary trees in real-world applications. A slightly more relaxed clustering algorithm that lets a larger number of labels to be clustered together per cluster may may be better suited to medline. The MeSH hierarchy based on 'human semantic', while achieving respectable performance, still failed to beat the best HAC HXML-CNN model as well as the baseline. Although the MeSH hierarchy had other issues complicating it's implementation, it appeared to have worked well in practice, just not as good as the baseline, and slightly worse than the artificially generated hierarchy (**RQ 3**).

The results of this project are not sufficient to conclude that the use of hierarchies is necessarily detrimental to the subject prediction task. Similar approaches have had success on other XML

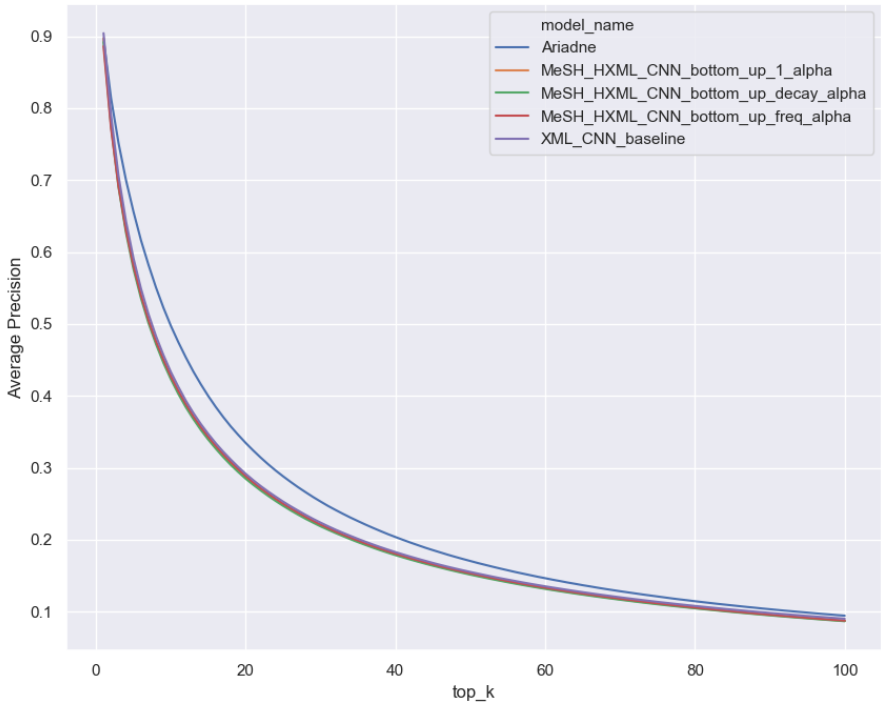


Fig. 14. Average Precision@K (for K=1, 2,...., 100) for the significant MeSH models trained on the medline dataset

datasets[48]. However, the challenges of medline such as being a domain-specific dataset with arcane terms, the presence of duplicate nodes with a lack of clarity in reverse engineering the correct position of annotated nodes, the use of subheadings as a secondary term to be predicted, etc., make it a challenging dataset, even amongst other XML datasets.

6.2 Duplicate MeSH Nodes

One of the main issues in experimenting with the MeSH hierarchy is the presence of duplicate nodes in the MeSH tree. While these nodes are intended to help NLM indexers to infer different meanings based on their position in the hierarchy, medline records do not have sufficient information to reverse-engineer the hierarchical position of an annotated label. Predicting the correct position of a label from its duplicate nodes could be a challenging task in its own right.

6.3 Penalties and Evaluation

Another issue is the class imbalance due to the power law distribution. Both the original XML-CNN and the HXML-CNN model variants were prone to the influence of head labels that dominate due to their large number of occurrences. This leads the model to predict the more frequent head labels confidently at the expense of the less frequent labels. Prior experiments with using weighted loss

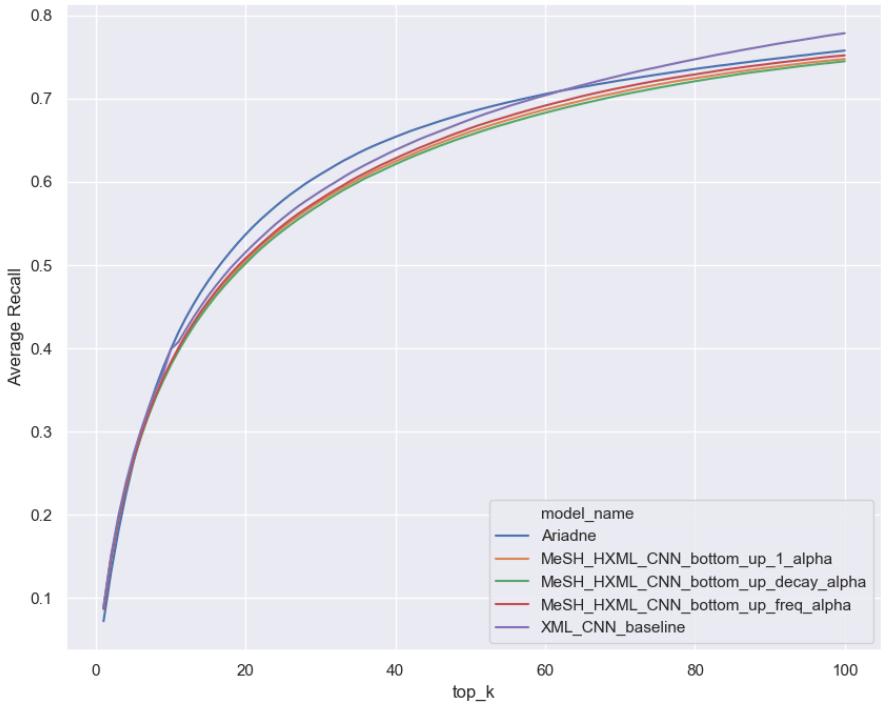


Fig. 15. Average Recall@K (for K=1, 2,...., 100) for the significant MeSH models trained on the medline dataset

functions to counter the imbalance has shown promise in limiting the head label influence and boosting the performance of the mid and tail labels. By penalising all output nodes individually, based on their training frequencies might provide an effective way to curtail the influence of head labels.

When penalising a loss function, whether by grouping output nodes as a form of MTL or individually as a single weighted loss function , there is a risk that of negatively affecting the performance metric, even if the tail label prediction is improved. This is due to the dependence of ranking evaluations on predicting the annotated labels correctly, which are dominated by the head labels. If the head labels are penalised too heavily and predicted less, the evaluation metric will suffer even if more relevant and specific tail labels are predicted. In order to perfectly balance the penalties and correct prediction of head labels, it may be more appealing to use a weighted loss function that penalises each output node separately.

6.4 Confidence Cut-Off

The current evaluation method for all deep learning models, including the baseline, involves ranking the predicted labels in the descending order of their confidence and choosing the top-k labels for evaluation. A more common approach is to set a cut-off limit on the confidence and only accept labels whose confidence values are higher than the cut-off. Usually, this cut-off is set to 0.5 in

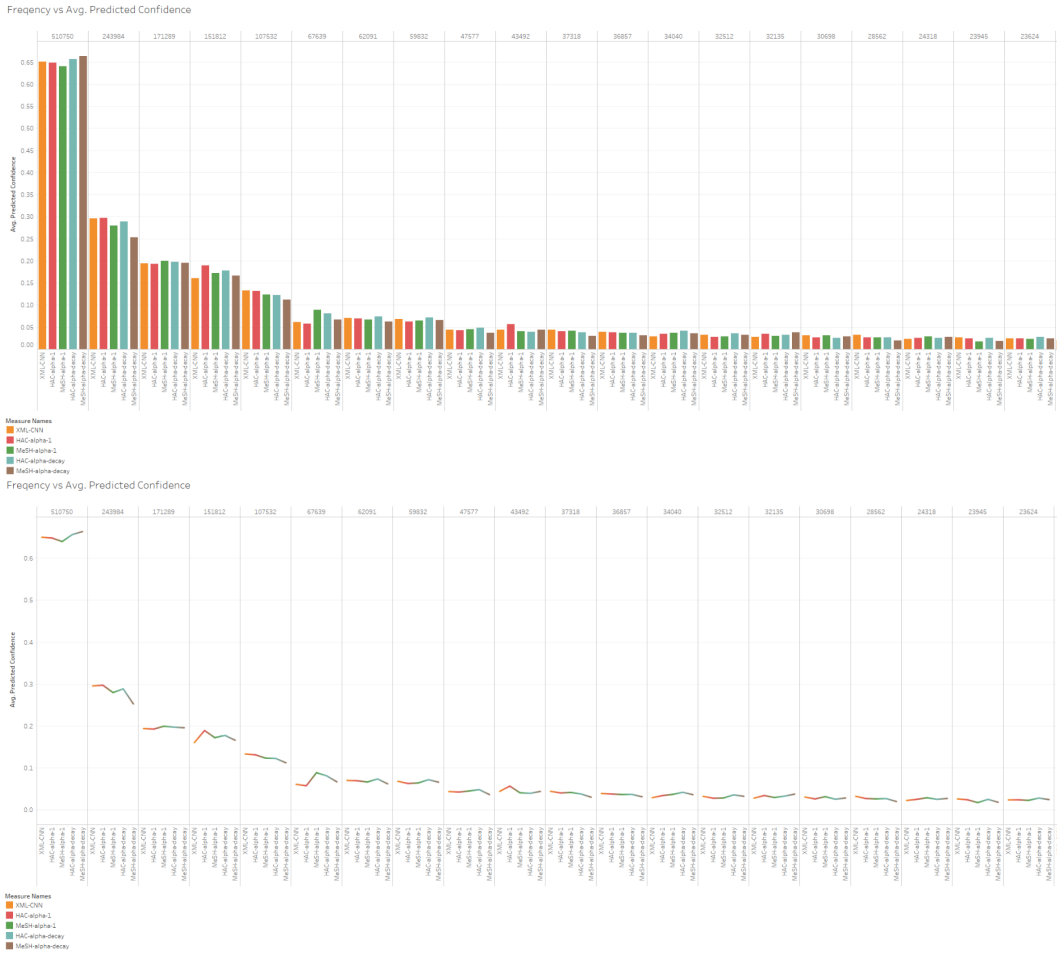


Fig. 16. Frequency vs Avg. Predicted confidence per label. The long tail end of the plot is not shown.

traditional classification tasks, considering it to be the probability that the label is correct. While this makes sense for these tasks, this approach would result in extremely poor performance for the medline dataset. The average predicted confidence for the top 100 labels of all models tested (including XML-CNN), was found to be extremely low, at 0.06. If we assume that in practical cases, only the top 20 predicted labels are to be considered, the average predicted confidence would still be much lower than 0.5, at 0.22. However, these low confidences cannot be necessarily ignored as extreme classification tends to produce such low confidences due to the power law distribution of the training samples available for labels. This is seen in fig. 16, where the confidence drops sharply as the frequency drops. The line plot shows a power law distribution emerge, mimicking the label frequency plot in fig. 3.

Additionally, if a cut-off of 0.5 is implemented, only an average of 3.17 labels are predicted per record, which is significantly lower than the average label density of 17 per record. Most of these labels would likely only be the most frequent labels, as is apparent from fig 16. This would sharply reduce the performance and usability of all models.

6.5 Evaluating under an incomplete ground truth

While ranking evaluation measures are well suited for XMTC tasks, there are fundamental problems in using such measures. Average precision and recall @k work by exclusively focusing on whether the predicted label is in the list of actual labels. There is no way for the current evaluation metric to identify a potentially semantically-relevant label (according to human judgement) that might not be in the actual list, even if it might be more relevant and/or useful than other predictions that may be in the list. Therefore, if a semantically relevant label is predicted by the model but isn't in the actual labels list, the performance suffers needlessly. For example, for the article in figure 1, the terms predicted by XML-CNN also included the terms *visual perception physiology, motion perception physiology, space perception physiology, and saccades*, all terms that appear to be relevant to the actual text, but are absent from the original list of MeSH terms. Even though head labels are essential to scoring high on the chosen performance metric, they are practically less useful to end-users compared to mid or tail labels. Considering an information retrieval task where medline records are to be retrieved using their MeSH headings, searching for a record using a head label will result in a large list of documents that the user then has to search through again to find the most relevant result. This is not desirable and is solved by the use of rare and specific terms.

Evaluation for XMTC problems will always suffer from this *'incomplete ground truth'* problem. Since only a handful of labels are annotated for each record, the list of possibly correct labels is never complete for any record. Hence, it is possible that a model predicts a semantically more relevant label for a record, but gets penalised if that label is not in the annotated list of labels for the record. This means that all evaluations of all XMTC problems are biased towards the list of annotated labels for each record, and that *true evaluation* (based on semantic relevance), is an open research question in the field of XMTC.

7 CONCLUSION AND FUTURE WORK

This project researched the use of Hierarchical Deep Neural Networks for MeSH subject prediction on the Medline dataset. The XML-CNN model architecture was implemented as a baseline for performance comparisons. Two hierarchies, one automatically generated using hierarchical agglomerative clustering and one manually created and maintained by NLM, were used to implement a hierarchical model based on the XML-CNN architecture. Each of these two models were trained with varying penalties (no penalty, steadily decayed penalty, and penalty based on frequency) to minimise the highly frequent head labels and ancestor nodes of specific labels from dominating predictions. It was found that while the performance of these models was comparable to the baseline, none of them managed to convincingly beat the baseline performance. Additionally, the Ariadne method for subject prediction implemented by OCLC manages to convincingly beat the baseline and does so with much less time required for training and prediction, all desirable qualities in an extreme classification task. The results for the hierarchical models are not sufficiently low to conclusively say that the hierarchical deep learning methods are ineffective for MeSH subject prediction.

For future research, it would be interesting to explore other hierarchy generation methods than HAC and find hierarchies that might be better suited to medline. One of the prime differences between the MeSH tree and the dendrogram generated by HAC is the fact that the latter enforced a binary tree, while the manual MeSH clusters had no such restriction on how many labels could be part of any given cluster. A clustering algorithm that loosens this constraint could produce different results, one that accommodates a more generous spread of labels within hierarchies such as the balanced k-means clustering algorithm. The prediction of subheadings is an auxiliary task that was ignored in this project for simplicity. However, it might be worthwhile to employ Multi-Task Learning and consider subheading prediction to be a better auxiliary task for hard parameter sharing. Finally, the use of a weighted loss function that penalises or aids label prediction based on the training sample frequency of each output node could show a potential improvement in performance if the penalties applied aren't too severe on the head labels. Further research exploring more hierarchies, penalisation and fine tuning methods are necessary to examine potential improvements in performance that hierarchical XML-CNN model can achieve over the regular XML-CNN model.

REFERENCES

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

[2] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 13–24.

[3] Shervine Amidi and Afshine Amidi. [n. d.]. A detailed example of data generators with Keras. https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly. ([n. d.]).

[4] Trent B. [n. d.]. Iterative-stratification: Scikit-learn cross validators for iterative stratification of multilabel data. https://github.com/trent-b/iterative-stratification. ([n. d.]).

[5] 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.

[6] Simon Baker and Anna Korhonen. 2017. Initializing neural networks for hierarchical multi-label text classification. *BioNLP 2017* (2017), 307–315.

[7] Jonathan Baxter. 1997. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning* 28, 1 (1997), 7–39.

[8] Samy Bengio, Jason Weston, and David Grangier. 2010. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*. 163–171.

[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] Wei Bi and James T Kwok. 2011. Multi-label classification on tree-and dag-structured hierarchies. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 17–24.

[11] R Caruana. 1997. Multitask learning: A knowledge-based source of inductive bias. Machine Learning. (1997).

[12] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. 2006. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research* 7, Jan (2006), 31–54.

[13] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter* 6, 1 (2004), 1–6.

[14] François Chollet et al. 2015. Keras. https://keras.io. (2015).

[15] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. 2013. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*. 1851–1859.

[16] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.

[17] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8599–8603.

[18] Emily Denton, Jason Weston, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. 2015. User conditional hashtag prediction for images. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 1731–1740.

[19] Jonathan P DeShazo, Donna L LaVallie, and Fredric M Wolf. 2009. Publication trends in the medical informatics literature: 20 years of" Medical Informatics" in MeSH. *BMC medical informatics and decision making* 9, 1 (2009), 7.

[20] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[21] Daniel J Hsu, Sham M Kakade, John Langford, and Tong Zhang. 2009. Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*. 772–780.

[22] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 935–944.

[23] Arif E Jinha. 2010. Article 50 million: an estimate of the number of scholarly articles in existence. *Learned Publishing* 23, 3 (2010), 258–263.

[24] R Johnson, A Watkinson, and M Mabe. 2018. The STM Report: an overview of scientific and scholarly publishing. (2018).

[25] Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. 2012. Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems*. 2645–2653.

[26] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).
[27] R. Koopman, S. Wang, and G. Englebienne. 2019. Fast and discriminative semantic embedding. In *Proceedings of the 13th International Conference on Computational Semantics (IWCS 2019)*.
[28] M Krendzelak and F Jakab. 2018. Approach for Hierarchical Global All-In Classification with application of Convolutional Neural Networks. In *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, 317–322.
[29] Gakuto Kurata, Bing Xiang, and Bowen Zhou. 2016. Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 521–526.
[30] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. 2007. Hierarchical document classification using automatically generated hierarchy. *Journal of Intelligent Information Systems* 29, 2 (2007), 211–230.
[31] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).
[32] 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*. ACM, 115–124.
[33] Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*. 1081–1088.
[34] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model.. In *Aistats*, Vol. 5. Citeseer, 246–252.
[35] U.S National Library of Medicine. [n. d.]. MeSH Browser. https://meshb.nlm.nih.gov/treeView. ([n. d.]). (Accessed on 03/18/2019).
[36] U.S National Library of Medicine. [n. d.]. Principles of MEDLINE Subject Indexing. https://www.nlm.nih.gov/bsd/disted/meshtutorial/principlesofmedlinesubjectindexing/. ([n. d.]). (Accessed on 03/18/2019).
[37] U.S. National Library of Medicine. 2019. Download MEDLINE/PubMed Data. (2019). https://www.nlm.nih.gov/databases/download/pubmed_medline.html
[38] Yannis Papanikolaou, Grigorios Tsoumakas, and Ioannis Katakis. 2018. Hierarchical partitioning of the output space in multi-label data. *Data & Knowledge Engineering* 116 (2018), 42–60.
[39] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[40] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 993–1002.
[41] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 993–1002.
[42] Yashoteja Prabhu and Manik Varma. 2014. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 263–272.
[43] 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*. ACM, 258–267.
[44] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. 2006. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research* 7, Jul (2006), 1601–1626.
[45] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).
[46] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2011. On the stratification of multi-label data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 145–158.
[47] Dan Shen, Jean David Ruvini, Manas Somaiya, and Neel Sundaresan. 2011. Item categorization in the e-commerce domain. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 1921–1924.
[48] Kazuya Shimura, Jiyi Li, and Fumiyo Fukumoto. 2018. HFT-CNN: Learning Hierarchical Category Structure for Multi-label Short Text Categorization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 811–816.
[49] Gerasimos Spanakis, Georgios Siolas, and Andreas Stafylopatis. 2012. Exploiting Wikipedia knowledge for conceptual hierarchical clustering of documents. *Comput. J.* 55, 3 (2012), 299–312.

[50] Yukihiro Tagami. 2017. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 455–464.
[51] Farbound Tai and Hsuan-Tien Lin. 2012. Multilabel classification with principal label space transformation. *Neural Computation* 24, 9 (2012), 2508–2542.
[52] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688
[53] Manik Varma. [n. d.]. Multi-label Datasets & Code. ([n. d.]). http://manikvarma.org/downloads/XC/XMLRepository.html
[54] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label partitioning for sublinear ranking. (2013).
[55] Ian EH Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. 2017. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 545–553.
[56] Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit S Dhillon. 2016. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification.. In *ICML*. 3069–3077.
[57] Ronghui You, Suyang Dai, Zihan Zhang, Hiroshi Mamitsuka, and Shanfeng Zhu. 2018. AttentionXML: Extreme Multi-Label Text Classification with Multi-Label Attention Based Recurrent Neural Networks. *arXiv preprint arXiv:1811.01727* (2018).
[58] Wenjie Zhang, Junchi Yan, Xiangfeng Wang, and Hongyuan Zha. 2018. Deep extreme multi-label learning. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ACM, 100–107.

A APPENDIX

A.1 Libraries and Software

The web urls for the different libraries and software used for the project are given below in table 2.

Software		**URL**
PyCharm	-	https://www.jetbrains.com/pycharm/download/
Git	-	https://git-scm.com/downloads
SourceTree	-	https://www.sourcetreeapp.com/
Anaconda	-	https://www.anaconda.com/distribution/
Nvidia Cuda	-	https://developer.nvidia.com/cuda-downloads
Nvidia cuDNN	-	https://developer.nvidia.com/cudnn
Libraries		**URL**
Tensorflow	-	https://www.tensorflow.org/install/pip
NumPy	-	https://github.com/numpy/numpy
Pandas	-	https://github.com/pandas-dev/pandas
Scikit-Learn	-	https://github.com/scikit-learn/scikit-learn
Gensim	-	https://github.com/RaRe-Technologies/gensim
Seaborn	-	https://github.com/mwaskom/seaborn
h5py	-	https://github.com/h5py/h5py
iterative-stratification	-	https://github.com/trent-b/iterative-stratification

Table 2. Urls for all Software and Python libraries used in the project

A.2 Experimental Parameters

Most parameters of the model were kept to the same specification in the original XML-CNN paper. Some were modified to better fit the medline dataset (embedding dimension and maximum input sequence length). These values are:

(1) Number of Filters: 32 (from XML-CNN)
(2) Number of Pooling Units: 32 (from XML-CNN)
(3) Filter Sizes: [2, 4, 8] (from XML-CNN)
(4) Vocabulary Size: 50,000 (from XML-CNN)
(5) Maximum Vocabulary for Embeddings: 50000 (from XML-CNN)
(6) Embedding Dimension: 256
(7) Maximum Input Sequence Length: 2048 (calculated as the power of 2 closest to the mean length of document size in the corpus)
(8) Dropout: 0.5
(9) Batch Size: 64