Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019)

Multi-Label Multi-Class Hierarchical Classification using Convolutional Seq2Seq

Venkatesh Umaashankar
Ericsson Research / Chennai
venkatesh.u@ericsson.com

Girish Shanmugam S
Intern, Ericsson Research / Chennai
s.girishshanmugam@gmail.com

Abstract

In this paper, We describe our approach for Germeval 2019 Task 1, a hierarchical multi-label multi-class text classification task. This task involves two subtasks where short descriptive text about German books need to be classified into one or multiple **(a)** top level categories (8 classes). **(b)** specific categories (343 classes). We present a novel approach of using Convolutional Seq2Seq modeling for solving both the tasks with a single model. In addition, We use category based random over sampling to handle the imbalance. Our approach reaches f1-micro score of **0.867** on ***Subtask (a)*** and **0.6722** on ***Subtask (b)***. Our approach achieved first rank in ***Subtask (a)*** and second rank in ***Subtask (b)*** in the test phase of the shared task. Our code is available in the link https://gitlab.com/vumaasha/germeval.

1 Introduction

Multi-label Multi-class Hierarchical classification (MLMCHC) refers to a setting where We can assign one or more labels to each instance (multi-label) where each label can have more than two possible classes (multi-class) that could be organized in a hierarchical structure (hierarchical). MLMCHC problems are common in domains like text classification (Rousu et al. (2006)), image classification (Hsu et al. (2009)) and bioinformatics (Barutcuoglu et al. (2006), Feng et al. (2017)). It is more commonly used in the field of Natural Language Processing (NLP) to classify text documents where a document can have multiple topics associated with them. Unlike the traditional flat classification approach, in MLMCHC the label cardinality (Charte et al., 2015) and number of labels is typically high. Also, the labels are interdependent and their distribution is skewed.

Traditionally, the hierarchical classification problem is solved by a binary relevance approach where the task is reduced to a flat classification problem by ignoring the label hierarchy and learning an independent binary classifier for each label in the taxonomy or ontology (Tsoumakas et al. (2009)). However, this approach neglects the correlations between labels. Cerri et al. (2016) follow a top-down strategy using neural networks where they use the previous level along with the feature vectors to predict the current level. The issue in this strategy is that the error in a level gets propagated to all the levels following it. Classifier chains (CC) proposed by Read et al. (2011) uses a chain of binary classification problems to model the correlations between labels. This approach is computationally expensive since it relies on training a cascade of classifiers.

Seq2Seq models have achieved tremendous success in machine translation (Bahdanau et al. (2014), Cho et al. (2014)). Li et al. (2018) and Hiramatsu and Wakabayashi (2018) have used RNN in their Seq2Seq models for product taxonomy classification. For machine translation tasks, RNNs are most preferred choice than CNN because of their superior performance on text applications. Gehring et al. (2017) have proposed a Convolutional Seq2Seq model which achieves state-of-the-art accuracy at nine times the speed of recurrent neural systems.

In our approach, we use ***Convolutional Seq2Seq architecture*** to model MLMCHC as a translation task, apply it to Germeval Task1 and evaluate the results. Experiments show that our approach can classify the books more precisely and our model reaches the f1-micro scores of $\mathbf{0.867}$ on ***Subtask (a)*** and $\mathbf{0.6722}$ on ***Subtask (b)***. Our approach achieved first rank in ***Subtask (a)*** and second rank in ***Subtask (b)*** in the test phase of the shared task. The rest of the paper is organized as follows. We describe the characteristics of the dataset in Section. 2. In Section. 3, we present our modeling pipeline that explains the sequence of steps in our approach. Feature engineering, imbalance handling and model architecture are explained in Section. 3.1, Section. 3.3 and Section. 3.6 respectively. In Section. 3.9 and Section. 4 we provide our experiment setup. Finally, in Section. 5 we conclude and provide details about the possible future works.

2 Germeval 2019 Task 1 Dataset

The dataset contains the attributes URL, ISBN, title, authors, blurbs, categories, and date of publication corresponding to German books which are crawled from randomhouse.de. The categories could be organized as a hierarchy tree and the metadata corresponding to the hierarchy is provided. This dataset follows the policies described in the RCV1 dataset by Lewis et al. (2004).

343 unique categories are hierarchically structured (8, 93 and 242 on level 1, 2 and 3 respectively). One or more specific categories are assigned to each book. Specific categories need not have to be a leaf node. For instance, the most specific category of a book could be *Romane & Erzäulungen*, although *Roman & Erzählungen* has further children categories, such as Romanbiographien.

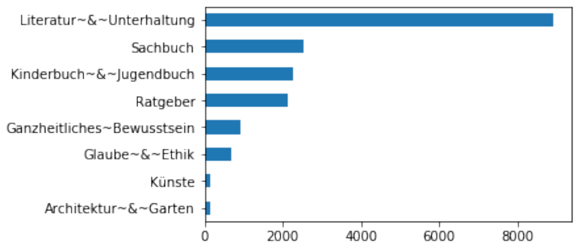

Figure 1: Top Level label distribution

The category distribution for top levels, all levels and specific categories are shown in figures 1, 3 and 2. From the distributions, it can be observed that label distributions on the top level and all levels are much skewed than that of specific categories.

Charte et al. (2015) provides various metrics for characterizing imbalance in Multi-Label Datasets (MLD). According to them, any MLD with a *Max Imbalance Ratio per Label (MeanIR)* value higher than 1.5 (50% more of samples with majority label vs minority label, in average) and *Coefficient of variation of IRLbl (CVIR)* value above 0.2 (20% of variance in the IRLbl values) should be considered

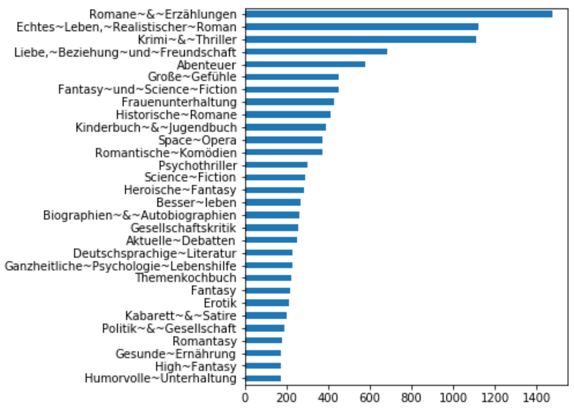

Figure 2: Label distribution for specific categories excluding ancestors for top 30 classes

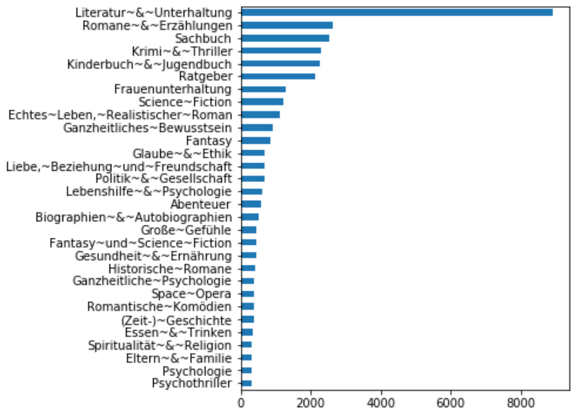

Figure 3: Label distribution including ancestors for top 30 classes

as imbalanced. Our dataset has a high *MeanIR* of **118.46** and high *CVIR* of **1.76** for specific category assignments. This shows that this dataset suffers from severe imbalance.

3 Modeling Pipeline

Our modeling work flow is shown in the figure 4. we used a single model for both the subtasks. Our model was trained to perform the ***Subtask (b)***. The results from the ***Subtask (b)*** are used to generate the results for ***Subtask (a)***.

3.1 Data Preparation

We created a hierarchy object from the relationship information provided in the file *hierarchy.txt*. This hierarchy object provides a programmatic interface to get information about any category in the hierarchy. The raw data is provided in the **XML** format. We parsed the XML files using *BeautifulSoup*, a Python package for parsing HTML and XML documents and converted them

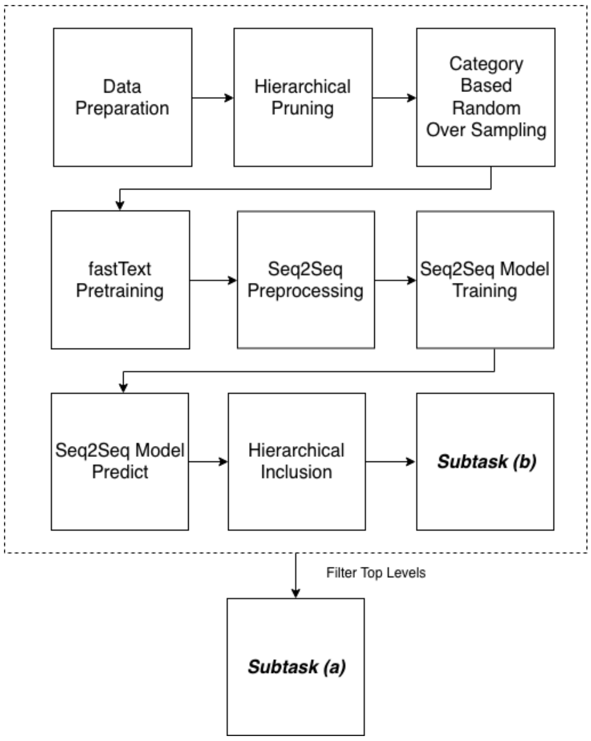

Figure 4: Modeling Pipeline

into **CSV** format.

We concatenated ***author, title, year*** along with the blurb into a single text for every book. For the author, title and year, we applied contextual concatenation by inserting markers at the start and end of the context. We concatenated the author information as (@AUTHOR <Author names> AUTHOR@) and similarly for title and year, we used (@TITLE <Title> TITLE@) and (@YEAR <Year> YEAR@). A 13 digit ***ISBN*** consists of five parts. We extracted the 2 and 3rd part corresponding to ***Group*** and ***Publisher*** and appended them to the input text as (@ISBN_GRP <Group> ISBN_GRP@ and @ISBN_PUB <Publisher> ISBN_PUB@). This contextual concatenation allows us to use a single shared embedding representation for multiple modalities such as ***author, title, year, ISBN*** without losing the context of individual attributes. Also, we preserve the punctuation and special characters by making them valid tokens. We split the labeled data into two splits for training (95%) and validation (5%) and use the unlabelled data as the test split.

3.2 Hierarchical Pruning

For each book, we only pick their specific categories (category tags that contain **label="True"** attribute) as labels. When a book has multiple specific categories, our label is a concatenated string of all the corresponding specific categories. After predicting these specific categories, the hierarchy object explained in Section. 3.1 can be used to query the ancestor categories of a predicted node, so we avoided including ancestor categories in our labels.

3.3 Category Based Random Over Sampling

In our approach, we only use specific category assignments as labels and skip the corresponding ancestors that can be looked using the hierarchy. Still, as we highlighted already in Section. 2 the training split suffers from severe imbalance. We alleviate this problem by performing category based random oversampling on the training split. Our oversampling algorithm is shown in Algorithm. 1. We oversample the training split by 15%, by using a value of $fraction = 0.15$. We observe improvements in the imbalance metrics, particularly the *MeanIR* reduces to 45.17 from 118.46. The imbalance metrics for specific categories on oversampled data is shown in Table 1 and a comparison of the distribution of top 30 minority classes are shown in the figure 5.

	Actual	Oversampled
Label Cardinality	1.46	**1.55**
Label Density	0.0043	**0.0045**
MeanIR	118.46	**45.17**
MaxIR	1474.00	1474.00
CVIR	**1.76**	1.85

Table 1: Imbalance characteristics for specific categories in Actual and Oversampled data

```
input : dataset, fraction
output: oversampled_dataset
oversample_size ← size_of (dataset) ∗ fraction;
category_wise_freq ← category_frequencies (dataset) ;
category_freq_mean ← mean (category_wise_freq);
minority_categories ← {};
foreach $category, freq$ ∈ category_wise_freq do
    if $freq$ < category_freq_mean then
        minority_categories ← minority_categories ∪ {$category$};
    end
end
// Average number of samples to be added for each
   minority category
mean_increment ← oversample_size/size_of (minority_categories) ;
over_samples ← {};
foreach $category$ ∈ minority_categories do
    mean_diff ← category_freq_mean − frequency (category) ;
    samples_to_add ← min (mean_diff, mean_increment) ;
    over_samples
      ← over_samples ∪ random_sample (category, samples_to_add) ;
end
oversampled_dataset ← dataset ∪ over_samples
```

Algorithm 1: Category Based ROS

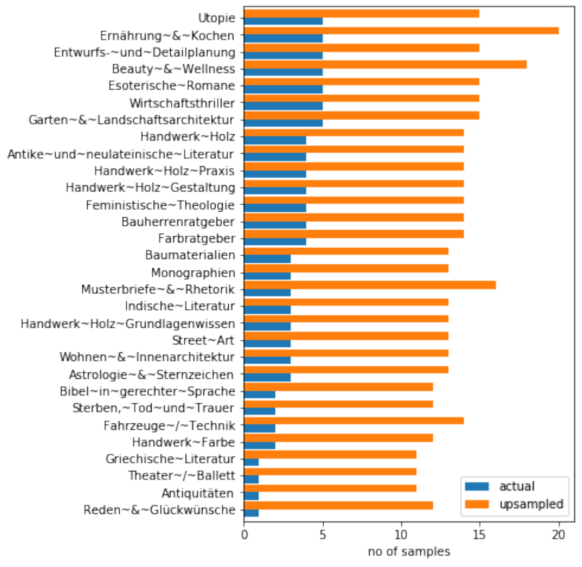

Figure 5: Top 30 Minority categories distribution

3.4 fastText Pretraining

In this phase, We use all the available data (oversampled training, validation and test splits) to learn fastText embeddings for all the tokens in the corpus. fastText provides an implementation for learning character n-grams based continuous word representations proposed by Bojanowski et al. (2017). Each word is considered as a bag of character n-grams and the representation for a word is obtained by the sum of the corresponding character n-gram embeddings. This approach has the advantage of taking morphological features of a word into consideration and also provides representations for words that are not seen in the training corpus.

We learn 2 sets of 100 dimensional embeddings. One for the tokens in the input text (blurbs, title, author, year and ISBN) and another for the categories. Each category in the hierarchy is considered as an individual token. We have a total of 142624 tokens in the input text and 343 tokens in the categories. We use skip-gram and negative sampling options available in fastText. We trained the word embeddings in the input text for 20 epochs and the category embeddings for 100 epochs with a learning rate of 0.05. We use these generic word embeddings to initialize the embeddings in our Seq2Seq model.

3.5 Seq2Seq Preprocessing

We use FAIRSEQ (Ott et al., 2019), a sequence modeling toolkit based on PyTorch. FAIRSEQ provides predefined architectures and components for Seq2Seq modeling. During preprocessing, FAIRSEQ uses our predefined vocabulary (a global dictionary of tokens from the whole corpus) and encodes the training, validation and test data into integers. The encoded data is saved data in a binary format that supports indexed access and faster loading time.

3.6 Seq2Seq Model Training

A Seq2Seq model mainly contains 2 components, namely *encoder* and *decoder*. The *encoder* converts the input sequence into a fixed size thought vector and the *decoder* sequentially generates the output sequence one step at a time by conditioning on the encoder output and predicted value in the previous time step. Recurrent Neural Networks (RNN) are a popular choice for solving Seq2Seq problems. However, their sequential nature implies that they take longer to train since the training cannot be parallelized.

Gehring et al. (2017) introduced a Seq2Seq architecture that is entirely based on Convolutional Neural Networks (CNN). Convolutional architecture reduces the training time significantly by allowing parallel computation across time and samples. The predictions have to be still performed sequentially, one step at a time. In this architecture, both *encoder* and *decoder* are made of convolutional blocks (figure 6). Each block contains one dimensional convolution with a kernel width k, which is followed by a *Gated Linear Unit* (GLU) (Dauphin et al., 2017) as non-linearity. The GLU facilitates the gradient propagation by implementing a simple gating mechanism over the convolution output. The GLU operation is given by the equation 1, where $Y = [A\ \ B] \in \mathbb{R}^{2d}$ is the convolution output $A, B, GLU([A\ \ B]) \in \mathbb{R}^{d}, \oplus$ is point-wise multiplication and σ is sigmoid operation.

$$GLU([A\ B]) = A \oplus \sigma(B) \qquad (1)$$

The convolutional blocks are stacked in multiple layers with residual connections from the input of the block to the output of the block to facilitate the flow of gradients during backpropagation. When the input and the output dimensions of block differ, linear projections are used in the residual connections to match the number of dimensions. Multi-layer CNN networks create hierarchical representations over the input sequence.

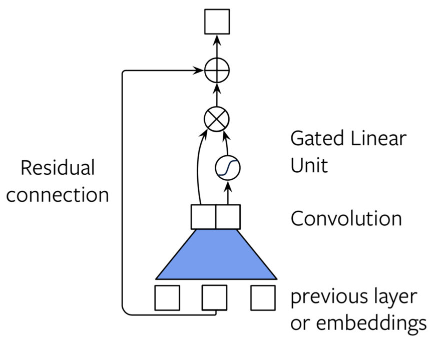

Figure 6: Convolutional Block

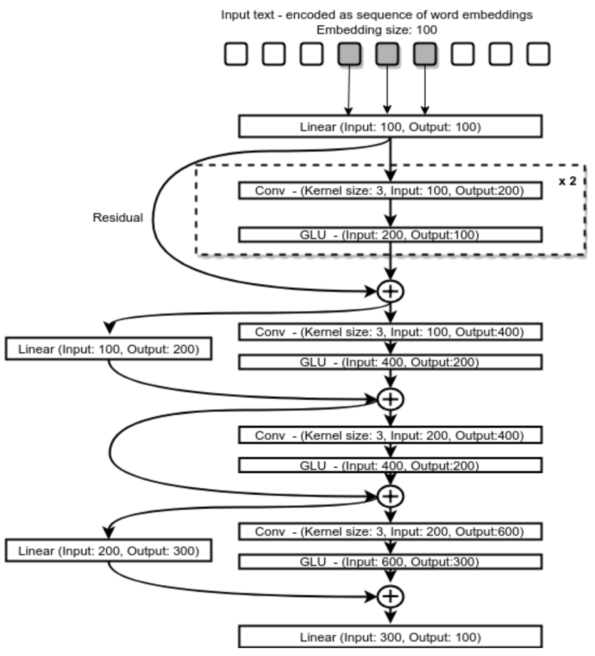
Input text - encoded as sequence of word embeddings
Embedding size: 100
Linear (Input: 100, Output: 100)
Residual
x 2
Conv - (Kernel size: 3, Input: 100, Output:200)
GLU - (Input: 200, Output:100)
Linear (Input: 100, Output: 200)
Conv - (Kernel size: 3, Input: 100, Output:400)
GLU - (Input: 400, Output:200)
Conv - (Kernel size: 3, Input: 200, Output:400)
GLU - (Input: 400, Output:200)
Linear (Input: 200, Output: 300)
Conv - (Kernel size: 3, Input: 200, Output:600)
GLU - (Input: 600, Output:300)
Linear (Input: 300, Output: 100)

Figure 7: Encoder

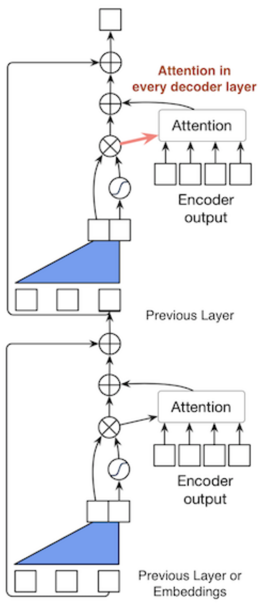
Attention in every decoder layer
Attention
Encoder output
Previous Layer
Attention
Encoder output
Previous Layer or Embeddings

Figure 8: Decoder with Multi-Hop Attention

This provides a quicker way to obtain dependencies between elements which are far apart in the sequence. With only $\mathcal{O}(\frac{n}{k})$ convolutional operations, the representations for n words in a sequence can be obtained, k is the kernel width. However, in a RNN, it would take $\mathcal{O}(n)$ linear operations to get the representation for n words. Further, an attention network is added to every layer in the decoder.

In our approach, We have modeled the MLMCHC as a Seq2Seq based translation task. We built a model that translates the given input text into a list of categories. Conceptually, this is similar to the Classifier Chaining since at each time step, We model the distribution $P(next\ category|previous\ categories, input\ text)$. However, our approach does not suffer from the problem of learning several classifiers.

3.6.1 Encoder

Our *encoder* (figure 7) starts with a linear layer of size (100×100) followed by 5 convolution blocks with output sizes $(100, 100, 200, 200, 300)$ and ends with a linear layer of size (300×100). All the convolutional blocks have a kernel width of 3. We experimented with different number of convolution blocks such 20,15,10,7 and 5. We choose to use 5 convolution blocks finally, since adding more number of blocks did not improve the validation f-score but increased the model complexity and size. The inputs to the encoder are the sum of the word and positional embeddings. Positional embeddings capture the ordering information by embedding the absolute position of the token in the input sequence.

3.6.2 Decoder

Our *decoder* architecture is similar to that of *encoder*. It starts with a linear layer of size (100×100) followed by 3 convolutions blocks with output sizes $(100, 100, 200)$, a linear layer of size (200×256) and ends with a linear layer of size (256×343) where 343 is the total number of categories in the hierarchy. Additionally, every *decoder* layer has an attention layer which allows the network to take repeated glimpses at the sequence and decide which input words are more relevant to predict the next word. At every decoder step, a decoder summary is calculated by combining the current decoder state with an embedding of the previous target element.

3.6.3 Multi-Hop Attention

We use multi-hop attention mechanism similar to Sukhbaatar et al. (2015) by carrying out this process for each step or hops. The attention for each source element is a dot product of the decoder summary and the output of the last encoder block. In multi-hop attention, the attention outputs for a layer is calculated based on the previous layer's attention results. Hence the decoder has access to attention values of all the previous layers which it uses to predict current layers output.

3.6.4 Model Parameters and Optimization

We initialize the word and category embeddings in the *encoder* and *decoder* using the fastText embeddings that We explained in the Section. 3.4 and fine-tune them as a part of training the Seq2Seq Model. We use Nesterov's accelerated gradient method with a momentum value of 0.99. We renormalize gradients if their norm exceeds 0.1. We use a fixed learning rate of 0.25. The hyper-parameters were chosen based on manual search. We trained the model for 13 epochs, after which the validation loss stopped improving. Our model has a total of 15962496 parameters and the size of our trained model is 122 MB.

3.7 Seq2Seq Model Predict

We use fairseq-generate to generate predictions using the Convolutional Seq2Seq model that We trained in the previous section. The predicted output is a sequence of all specific categories for the given input data. During the prediction phase, We use ***beam search*** with a beamwidth of 5 to identify the most probable output sequence.

3.8 Hierarchical Inclusion

We use hierarchy object introduced in the Section. 3.1 to query the ancestor categories of specific categories predicted by the Seq2Seq Model. This gives the solution for ***Subtask (b)***. We derive the solution for ***Subtask (a)*** from the solution for ***Subtask (b)*** by picking the corresponding top levels for predicted specific categories.

3.9 Experiment Setup

(1) Nvidia GPU GEFORCE GTX 1080 Ti 11GB RAM **(2)** Intel® Xeon® Processor E5-2650 v4 30M Cache, 2.20 GHz, 12 Cores, 24 Threads **(3)** 250 GB RAM **(4)** CentOS 7

4 Results

The test evaluation metrics are given in the Table. 2. In the test evaluation of the competition, Our model secured the first rank in ***Subtask (a)*** with a f1 score of $\mathbf{0.867}$ and second in ***Subtask (b)*** with a f1 score of $\mathbf{0.6722}$.

	Subtask (a)	***Subtask (b)***
Precision	0.8923	0.7377
Recall	0.8432	0.6174
F1-Score (micro)	0.867	0.6722
Accuracy	0.8364	0.3791

Table 2: Evaluation metrics on test data

5 Conclusion

In our solution, We have successfully demonstrated that Convolutional Seq2Seq modeling is a promising approach to address MLMCHC. We observed that the oversampling and pretraining phases were key ingredients of our successful recipe. In general, this emphasizes the importance of transfer learning in NLP problems. In the future, We plan to extend our approach by using sophisticated transformers based architecture for both pretraining and modeling phases.

6 Credits

The authors would like to thank their colleagues at Ericsson Research for their support and valuable inputs during this competition. Also, We would like to thank the Facebook research community for developing and maintaining the fastText and FAIRSEQ. Finally, We thank the GermEval competition organizers for fostering a friendly and collaborative environment around this dataset and for answering our questions throughout the competition.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Zafer Barutcuoglu, Robert E Schapire, and Olga G Troyanskaya. 2006. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ricardo Cerri, Rodrigo C Barros, André CPLF de Carvalho, and Yaochu Jin. 2016. Reduction strategies for hierarchical multi-label classification in protein function prediction. *BMC bioinformatics*, 17(1):373.

Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. 2015. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, 163:3–16.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941. JMLR. org.

Shou Feng, Ping Fu, and Wenbin Zheng. 2017. A hierarchical multi-label classification algorithm for gene function prediction. *Algorithms*, 10(4):138.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Makoto Hiramatsu and Kei Wakabayashi. 2018. Encoder-decoder neural networks for taxonomy classification.

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*, pages 772–780.

David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

Maggie Yundi Li, Liling Tan, Stanley Kok, and Ewa Szymanska. 2018. Unconstrained product categorization with sequence-to-sequence models.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2011. Classifier chains for multi-label classification. *Machine learning*, 85(3):333.

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):1601–1626.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2009. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer.