Neural Architecture Search for Extreme Multi-Label Text Classification

Loïc Pauletto, Massih-Reza Amini, Rohit Babbar, Nicolas Winckler

Neural Architecture Search for Extreme Multi-Label Text Classification

Loïc Pauletto[1,2] (✉), Massih-Reza Amini[2], Rohit Babbar[3], and Nicolas Winckler[1]

[1] Atos, Grenoble, France{loic.pauletto,nicolas.winckler}@atos.net
[2] University Grenoble Alpes, Grenoble, France
Massih-Reza.Amini@univ-grenoble-alpes.fr
[3] Aalto University, Helsinki, Finland rohit.babbar@aalto.fi

Abstract. Extreme classification and Neural Architecture Search (NAS) are research topics which have recently gained a lot of interest. While the former has been mainly motivated and applied in e-commerce and Natural Language Processing (NLP) applications, the NAS approach has been applied to a small variety of tasks, mainly in image processing. In this study, we extend the scope of NAS to the task of extreme multilabel classification (XMC). We propose a neuro-evolution approach, which was found to be the most suitable for a variety of tasks. Our NAS method automatically finds architectures that give competitive results with respect to the state of the art (and superior to other methods) with faster convergence. In addition, we perform analysis of the weights of the architecture blocks to provide insight into the importance of different operations that have been selected by the method.

Keywords: Neural architecture search · Machine Learning · Extreme classification · Evolutionary algorithms.

1 Introduction

Neural networks (NNs) have shown impressive performance in many natural language tasks, such as classification, generation, translation and many others. One of the applications that has attracted growing interest in recent years with the availability of large-scale textual data is the extreme multi-label text classification (XMC). The goal in XMC is to classify data to a small subset of relevant labels from a large set of all possible labels [13, 16]. A major problem in applying NNs to this task is to design an architecture that can effectively capture the semantics of text. Diverse methods have been employed in the NLP field, such as convolutional neural networks [25], recurrent neural networks [14] as well as a combination of both [26]. However, this design phase is complex and often requires human prior, with a good knowledge of the field and data. Over the last few years, NAS research has paved the way for the creation of dedicated neural architectures for a given task or even dataset. Most of the NAS studies, have focused on search algorithms for a small number of tasks (eg. image classification) and none of these studies have been applied to XMC before. In this

paper, we propose `XMC-NAS` a NAS-based method for automatically designing an architecture for the extreme multi-label text classification task, using a minimum of prior knowledge. In addition, we define a search space with operations (e.g. RNN, Convolution,..) specific to the NLP domain. To evaluate our solution, we have used 3 large scale XMC datasets with an increasing number of labels. Like popular NAS methods we have uses a proxy dataset to train and evaluate architectures during the search phase. The discovered architecture gives competitive results with respect to the state of the art on the proxy dataset with faster convergence. Then we transfer the best performing architecture to other datasets and evaluate it. Our evaluation shows, the discovered architecture also achieves results close to the state of the art. Furthermore, this paper presents a study on the importance of operation types and the network depth with respect to the obtained results.

In the following section, we briefly review some related state-of-the-art. In Section 3, we present our solution to extreme multi-label classification with neural architecture search. Experimental results are presented in Section 4 and the conclusion and an outcome of this work are presented in Section 5.

2 Related Work

In this section, we will present related work on neural architecture search and extreme text classification.

Neural Architecture Search. Studies on the subject of NAS date back to the 1990s [9] and they have gained significant interest in the last few years. In the literature, different approaches have been studied, one of the first approaches was based on Reinforcement Learning (RL) [27]. In these approaches, architectures are first sampled from a controller, typically a RNN, and are further trained and evaluated. The controller is updated from the evaluation results in a feedback loop, improving the sampled architectures over time. Some other approaches use Bayesian Optimization (BO) like [6] in order to predict the accuracy of a new and unseen network and thus select only the best operation or as in [11] which uses Sequential Model-Based Optimization (SMBO) to predict accuracy of a network based on a network with fewer operations. Transfer Learning [22] has also been used in NAS methods [18] allowing more efficient search by weight sharing of overlapping operations, instead of training each new network from scratch. Other NAS methods have used gradient descent based approaches such as in [12] where they use a relaxation, which allows to learn the architecture and the weight of operation simultaneously, using the gradient descent. More recent studies have shown great performances using the well-known evolution algorithms as in [15, 20], which consists in starting with a base population and successively apply mutation to the best performing architecture.

Extreme Text Classification. Different methods have been proposed to address the stakes posed by the extreme multi-label text classification [1, 2, 7]. The

most recent of those methods are deep learning based such as XML-CNN[13], a structure of convolutional neural network (CNN) and pooling in order to get a precise text representation. However, it is hard for CNN to capture the most relevant part of a text and the long term dependency. Other methods, more similar, to Seq2seq methods have been applied as discussed in MLC2Seq[16], SGM[23] and AttentionXML [24]. Those methods used recurrent neural network (RNN) to classify the text. Moreover, a significant interest has been given on attention mechanism the last few years[10]. Attention mechanism has demonstrated great performance in sequence modeling, in particular in NLP domain and has therefore been also applied in the context of XMC [23, 24].

3 Framework and Model

We propose `XMC-NAS`, a tool to automatically design architecture for the extreme multi-label classification task. Our approach is based on three main components: i) the text embedding, ii) the search of the architecture, and iii) output classification. These three components form a pipeline in which components i) and iii) are fixed and excluded from the search task. Thus, the architecture search task is performed only on the component ii). The first component of our methods consists in transforming the text into word embedding, i.e. numerical vectors. This embedding step should allow the model to use these representations to produce a more accurate prediction. The second step is the search phase for the most performing architecture, using an *evolutionary algorithm* (c.f. Fig.1). Finally, the last component classifies the output, in several categories. This last component is based on attention mechanism and fully connected layers.

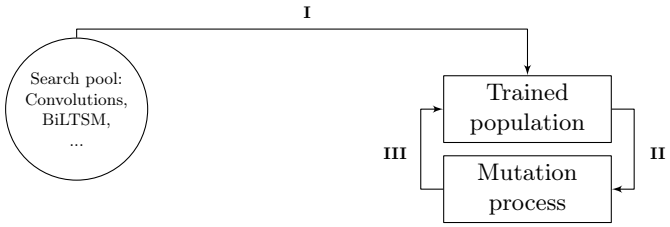


Fig. 1: **I**. Architectures are constructed from randomly sampled operations and then trained and evaluated, **II**. Randomly sample 10 architectures, and rank them by Precision@5 obtained on test set. The most performing one is selected for mutation, **III**. The newly mutated architecture, is trained and evaluated. Then placed in the trained population. The oldest architecture is removed from the population.

In the following section, we describe our approach in detail. First, we present the search space, the search algorithm used and their specificities; and finally, we describe the different parts of the discovered network.

3.1 Architecture search phase

The search phase can be broken down as follows. The architecture is searched in a *search space* that defines the possible structure of the final architecture. In this search space we have *candidate operations* that can be used in the architecture. Finally, to research the architecture, we use a *search algorithm* that searches for best architectures in the search space.

Search Space. A neural network architecture is represented in the form of a Direct Acyclic Graph (DAG). Each node in this graph represents a layer. A layer is a single operation, which is chosen among the set of *candidate operations*. The edges of the graph represent the data flow, and each node can have only one input. The graph is constructed as follows: first, the nodes are sampled sequentially (i.e. an operation is selected) to create a graph of N nodes. Then, the input of a node j is selected in the set of previous nodes (i.e. nodes from 1 to $j-1$). This set is initialized with a node that represents the embedding layer. Finally, when the node j has an operation and an input, it is added to the set of the previous nodes. To have a trade-off between performance and search efficiency, we have set a limit to the maximum number of previous nodes that a layer can take as input. We empirically determined this limit to 5 to achieve reasonable search time on our hardware. Figure 2 illustrates a simple architecture with $N = 6$ nodes (i.e layers).

Candidate Operations. To build our set of *candidate operations*, we have selected the most common operations in NLP field, which consist of a mixture of convolutional, pooling and recurrent layers. We have defined four variants of 1D convolutional layers, with a kernel of different size: 1, 3, 5 and 7 respectively. All convolution layers use a stride of 1 and use padding if necessary to keep a consistent shape. We used the two types of pooling layers that calculate either the average or the maximum on the filter size, this size is set to 3 for both. Similarly to the convolution layers, the pooling layers use a stride of 1 and use padding if necessary. Finally, we used the two popular types of recurrent layers, namely the Gated Recurrent Unit (GRU)[3] and the Long-Short Term Memory (LSTM)[4], which are able to capture long-term dependencies. Specifically, we use bi-directional LSTM and GRU.

Search Algorithm. As NAS algorithm we use the *regularized evolution* as described in [20]. We chose this approach because it allows us to have a fine vision of the impact of each operation on the final result. Regarding the mutation aspect, we use the same configuration as described in [20]:

- Randomly select an input from a node on the network and modify it with a new input.
- Randomly select an operation from the network and change it with a new sample.

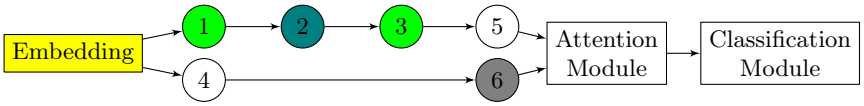


Fig. 2: Illustration of a simple architecture, with 6 layers. The numbers represent the sampling order of the layers. The limit of maximum number of previous layers that can be used as input is set to 5 (e.g. the layer 6 could hence take as input only nodes from 1 to 5). Here, different operations are illustrated with different colors.

Figure 1 illustrates the search algorithm of the *regularized evolution*. In order to see the impact of the number of layers on the final results, a third mutation, corresponding to the addition of a new layer, has been introduced. The choice among these mutations is random. We also seek to better understand how operations perform together, i.e. to evaluate the importance of the various operations with respect to the final results. To do this, we use a linear combination of outputs from each layer where weights are learned during the training process.

3.2 Text Embedding, Attention and Classification Modules

Our network has certain parts fixed, namely the embedding, attention and classification modules. This section will introduce them.

Text Embedding. The embedding layer produces a fixed length representation. This layer is an embedding map, which means each word is mapped to a vector. As initialization, we used the GloVe[17] embedding 840B-300d[1] version, which allows us to skip the step of learning a new embedding from scratch.

Attention Module. We use a self-attention mechanism based on the one demonstrated in [10], similarly to [24]. The attention process helps to grasp the important parts of the text. This mechanism uses a vector $\mathbf{c_t}$ that represents the relevant *context* for the label t, where t is in $1, \ldots, T$. For an input sequence of size N, the context vector is given in equation 1.

$$\mathbf{c_t} = \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h_i}, \qquad (1) \qquad\qquad \alpha_{t,i} = \frac{e^{\mathbf{W_t}^T \cdot \mathbf{h_i}}}{\sum_{k=1}^{N} e^{\mathbf{W_t}^T \cdot \mathbf{h_k}}} \qquad (2)$$

Where, $\mathbf{h_i}$ denotes the hidden representation,i.e. the output of RNN encoder states or of the convolution. In the case of BiRNN, layer $\mathbf{h_i}$ is the concatenation of vectors from the forward $\overrightarrow{h_i}$ and backward $\overleftarrow{h_i}$ passes. The term $\alpha_{t,i}$ is called attention factor (Eq. (2)). The set of attention factors $\{\alpha_{t,i}\}$ represents how much of each inputs should be considered for each output. In Eq. (2), $\mathbf{W_t}$ is the attention weight (i.e a learnable weight matrix) for the t-th label.

[1] `http://nlp.stanford.edu/data/wordvecs/glove.840B.300d.zip`

Classification Module. The final part of the network is composed of 2 or 3 fully connected layers, which reduces the output of the attention module. The result is then fed into an output layer that classifies it into different labels.

3.3 Analysis of Operations Importance

This section presents an analysis of the weights of the linear combination, particularly the impact of each operation on the final results, and whether different operations combine efficiently. We address this analysis in two steps. In the first step, we focus on how operations combine with each other. In a second step, we analyze the results and the impact of operations as the networks deepen.

First Step: In this step, the base population is randomly initialized, meaning that the input and operation of each node is chosen at random. We try to determine which operation is the most important in the first layers by scaling their outputs with trainable weights of the linear combination. The Fig.3 shows three examples of the first layers for different combinations of operations as well as the corresponding learned weights assigned to each operations. The block "Rest of the network" represents the attention and classification modules. The blocks in the hatched areas of the Fig.3 were not part of the mutation process and were "constrained". For each architecture example, the displayed weights are the averages obtained over several runs of the NAS. The different grey scales indicate different experiments. We observe in Fig.3 that pairs of operations of same type, i.e. BiLSTM, tend to have almost equal weights. However, some trends could be observed in the case of the combination of two convolutions; those with a larger kernel size have higher weights. This effect is particularly pronounced in the case of the kernel size of 1, reflecting the need for sequence modeling blocks at this level. In the case of mixed operations, it turned out that BiLSTM operations systematically have higher weights. An example of a run with mixed operation is presented in the right-hand side of Fig.3. More generally, our results show that architectures which contain BiLSTM at the first layer, perform better. This first step shows that the result is based mainly on the long-term dependencies captured by BiLSTM rather than on the combinations of local features generated by the convolution.

Second Step: This second stage of analysis aims to quantify the impact of the number of layers on the final results as well as the weights assigned to each operation. According to the results obtained in the first step, which show that the network with BiLSTM layers works better, in this second step, a part of the population has the constraint to start with at least one BiLSTM layer, which takes as input the embedding. For the analysis of the impact of the number of operations, we calculated the average $P@5$ based on the number of operations. The number of operations ranged from 2 to 6. We observed that the average precision is almost constant, regardless of the number of operations in the network, with a range of results close to what we have previously obtained. This result is corroborated by the analysis of the combination of operation weights. The Fig.4 shows examples of architecture for different combinations of operations with associated weights. This time the operations are assigned sequentially

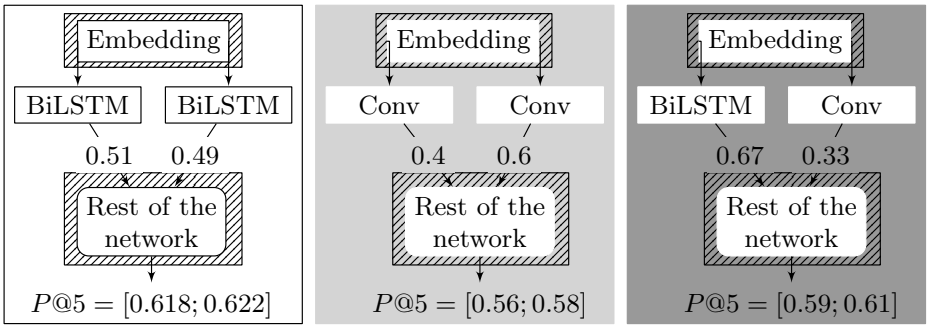


Fig. 3: Visualization of the network architecture with the applied weight on each operation. The weights have been averaged over multiple runs, the range of $P@5$ are obtained on the proxy dataset. For the central case, we also averaged over the kernel size.

(i.e. one after the other) forming a deeper network. The blocks in hatched areas in Fig.4 are partially or totally part of a constraint as in the previous subsection. Here, the blocks "Rest of the network" represent the following layers in the network, not shown for the sake of clarity, and still followed by the attention and classification modules. As previously, the weights displayed for each type of architecture are the averages obtained from multiple runs of the NAS. We note on the Fig.4 that the weights on additional layers are small compared to those that bypass it. This trend has been observed in all experiments and suggests that, given our operations pool, additional layers do not provide much more information. Thus, the information important for the result is extracted by layers that take the embedding as input.

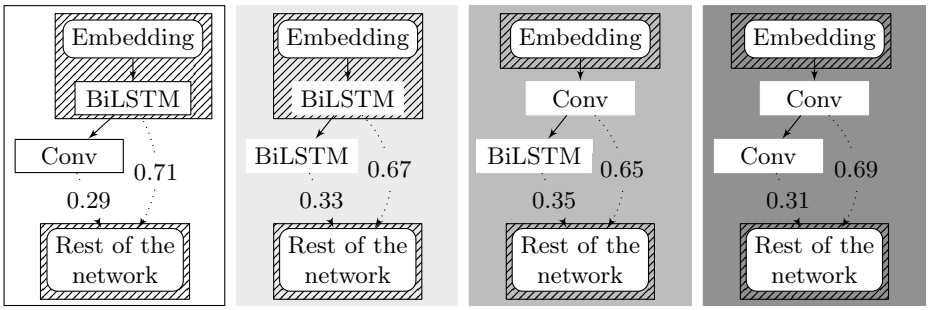


Fig. 4: Visualization of the network architecture, when the network is deeper. The dotted line represents the linear combination of all the layers outputs, with the weight applied on each outputs. The weight indicates that layers which take the embedding layer as input, have a predominant importance on the final result.

4 Experimental Results

We have conducted a number of experiments to evaluate how the proposed `XMC-NAS` method can help design an efficient neural network model for multi-label text classification.

4.1 Datasets and Evaluation Metrics

We conducted our study on three of the most popular XMC benchmark datasets downloaded from the XMC repository[2]. These datasets are considered large scale, with the number of class labels ranging from 4,000 to 30,000, which are listed from smallest to largest (in term of number of labels) by EURLex-4K, AmazonCat-13K, and Wiki10-31K summarized in Table 1. We followed the same pre-processing pipeline as the one used in [24]. To create the validation set we perform a split with the same initialization seed for all experiments. As evaluation metrics we used the Precision at k denoted by $P@k$, and the normalized discounted cumulative gain at k denoted by $nDCG@k$ [5]. Both metrics are standard and widely used in the state of the art references.

Table 1: Statistics of XMC datasets used in our experiments. L: # of classes.

Dataset	# of Training examples	# of Test examples	L	Avg. of class labels per example	Avg. size of classes
EURLex-4K	15,539	3,809	3,993	25.73	5.31
Wiki10-31k	14,146	6,616	30,938	8.52	18.64
AmazonCat-13K	1,186,239	306,782	13,330	448.57	5.04

4.2 Architecture Search Evaluation

This section presents the data and the hyperparameters that we have used during the search phase of our method. Finally, we present the most performing architecture that has been discovered on the proxy dataset.

Parameters and Data. We performed the search phase on the relatively small EURLex-4K dataset for scalability considerations, we call it the proxy dataset. In each experiment we create a base population of 20 networks. We then apply 50 rounds of mutations. For our experiments we have used the same hyperparameters as in [24], for the training of each sampled network. Namely, the optimizer was Adam [8] with a learning rate set to 0.001, and the maximum number of epochs were set to 30 epochs with early stopping. To be consistent with [24] we have used the same number of hidden states for the LSTM, which are specified in Table 2. The training stops if the performance of the network does not increase during 50 consecutive steps. We have used the cross-entropy loss function as proposed in [13] for training the models.

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

Table 2: Hyperparameters used for the training of the discovered model.

Dataset	Valid size	BiLSTM Hidden size	Fully connected
EURLex-4K	200	256	[512,256]
Wiki10-31k	200	256	[512,256]
AmazonCat-13K	4000	512	[1024,512,256]

The Discovered Architecture. The architecture found by `XMC-NAS`, consists of two BiLSTMs that take the same input and holds their own representation. The outputs of the two BiLSTMs is then concatenated along the hidden dimension, and given as the input of the self-attention block. Finally, we use a chain of fully connected networks to classify the sequence. For the training detail we use the same as presented in the previous paragraph (see also Table 2).
The architecture of the network discovered by `XMC-NAS` is presented in Fig.5.

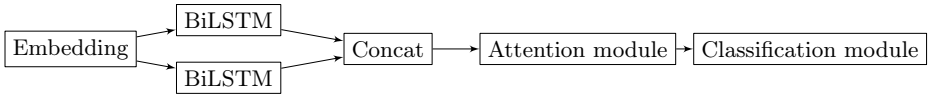


Fig. 5: The discovered network by `XMC-NAS`, is composed of two BiLSTM whose outputs are concatenated, and then passed in an attention module followed by fully connected layers.

4.3 Performance Evaluation

In this section we will present the results obtained by the `XMC-NAS` discovered architecture (Figure 5) on various XMC datasets (Table 1). First we present the results obtained on the proxy dataset (EURLex-4K) used for the search phase. Finally, we evaluate the performance of this discovered architecture, transferred on the other datasets. To train our network, we use 2 Nvidia GV100, with data parallelism training. The search time on the proxy dataset, depending on the configuration, ranges from 6 hours to 5 days. We compare the results of our method to the most representative methods on XMC (with the results provided by the authors in corresponding papers). Some of these techniques are deep learning based like MLC2Seq[16], XML-CNN[13], Attention-XML[24]. The others techniques are, AnnexML[21], DiSMEC[1], PfastreXML[5] and Parabel[19].

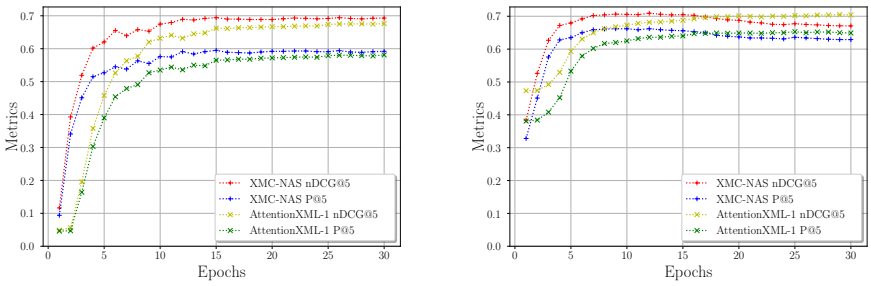
EURLex-4K Results. As presented in the left-hand side of the Table 3 we have obtained an improvement on P@1, P@3 and P@5 with respect to the state of the art. The shown precisions are the average over 3 different initializations. A significant improvement is obtained on the precision at 3 and 5, where we obtain 0.738 and 0.620 respectively compared to 0.730 and 0.611 before. The Fig.6a presents the evolution of P@5 and the nDCG@5 over the validation set with

Table 3: Comparison performance table on three datasets. Our methods surpass the state of the art in 4 cases and get competitive results that really close to the state of the art otherwise.

	EURLex-4K			Wiki10-31K			AmazonCat-13K		
Methods	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
AnnexML	0.796	0.649	0.535	0.864	0.742	0.642	0.935	0.783	0.633
DiSMEC	0.832	0.703	0.587	0.841	0.747	0.659	0.938	0.791	0.640
PfastreXML	0.731	0.601	0.505	0.835	0.686	0.591	0.917	0.779	0.636
Parabel	0.821	0.689	0.579	0.841	0.724	0.633	0.930	0.791	0.645
XML-CNN	0.753	0.601	0.492	0.814	0.662	0.561	0.932	0.770	0.614
AttentionXML-1	0.854	0.730	0.611	**0.870**	**0.777**	0.678	**0.956**	**0.819**	**0.669**
`XMC-NAS`	**0.858**	**0.738**	**0.620**	0.849	0.772	**0.681**	0.951	0.813	0.664

respect to the number of epochs. We can observe that our network has a faster convergence. The results is obtained around 15 epochs and after this point, the improvement is relatively small, which indicates that the network might overfit. It is not impossible that the improvement obtained is due to a larger network. However, we have systematically observed faster convergence in all the cases we have experienced. Furthermore, the contribution of the embedding or attention module on the results is not yet clear, as we have not yet studied the impact of these modules.

Architecture Transfer Results. We train and evaluate the discovered architecture following the same training procedure as defined in section 4.2 and using the hyperparameters presented in Table 2 on the two other datasets. The middle and right side of Table 3 show the comparison of the architecture dis-



(a) EURLex-4K (b) Wiki10-31K

Fig. 6: Plot of the nDCG@5 and P@5 on the validation set, on two different datasets. We notice, the discovered architecture have a faster convergence compared to the current state of the art. In the 6a our method get better final results, in 6b our final results (around epoch 15) are close.

covered by `XMC-NAS` on EURLex-4K with others methods. We notice that the best discovered architecture transferred to larger datasets obtains results close to the current state of the art. In some cases we slightly exceed the results as in the case of $P@5$ on the Wiki10-31K. Moreover we notice in Fig.6b that our methods still have a faster convergence, the same trend as observed on proxy dataset (cf. Fig.6a). Moreover, our results on Wiki10-31K and AmazonCat-13K are obtained in half of the epochs required by AttentionXML-1.

5 Summary & Outlook

We have presented in this work an automated method to discover architecture for the specific task of extreme multi-label classification, based on the regularized evolution [20] and with a domain oriented pool of operations. This method has found architectures that have provided competitive results with the existing state of the art methods [24], and in some cases overpassed them. Moreover, our method showed faster convergence rates on all datasets, which are more likely due to a higher complexity of the model. In addition, trainable weights were introduced on each operation of the pool in order to provide more understanding on the impact of each architecture blocks. Many directions are possible as future steps. This includes the tuning of the various hyperparameters, the study of the impact of attention and embedding modules, the development of a method that can handle the scale of datasets, and speed up the search process (e.g. using transfer learning).

References

1. Babbar, R., Schölkopf, B.: Dismec: Distributed sparse machines for extreme multi-label classification. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. pp. 721–729 (2017)
2. Babbar, R., Schölkopf, B.: Data scarcity, robustness and extreme multi-label classification. Machine Learning **108**(8-9), 1329–1351 (2019)
3. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
5. Jain, H., Prabhu, Y., Varma, M.: Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In: Proceedings of the 22nd ACM SIGKDD ICKDD. pp. 935–944 (2016)
6. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD ICKDD. pp. 1946–1956 (2019)
7. Khandagale, S., Xiao, H., Babbar, R.: Bonsai: diverse and shallow trees for extreme multi-label classification. Machine Learning pp. 1–21 (2020)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
9. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Systems **4** (1990)

10. Lin, Z., Feng, M., Santos, C.N.d., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding. arXiv preprint arXiv:1703.03130 (2017)
11. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
12. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
13. Liu, J., Chang, W.C., Wu, Y., Yang, Y.: Deep learning for extreme multi-label text classification. In: Proceedings of the 40th International ACM SIGIR. pp. 115–124 (2017)
14. Liu, X., Gao, J., He, X., Deng, L., Duh, K., Wang, Y.Y.: Representation learning using multi-task deep neural networks for semantic classification and information retrieval (2015)
15. Maziarz, K., Tan, M., Khorlin, A., Chang, K.Y.S., Jastrzębski, S., de Laroussilhe, Q., Gesmundo, A.: Evolutionary-neural hybrid agents for architecture search. arXiv preprint arXiv:1811.09828 (2018)
16. Nam, J., Mencía, E.L., Kim, H.J., Fürnkranz, J.: Maximizing subset accuracy with recurrent neural networks in multi-label classification. In: Advances in neural information processing systems. pp. 5413–5423 (2017)
17. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
18. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
19. Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., Varma, M.: Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In: Proceedings of the 2018 World Wide Web Conference. pp. 993–1002 (2018)
20. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
21. Tagami, Y.: 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. pp. 455–464 (2017)
22. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: International conference on artificial neural networks. pp. 270–279. Springer (2018)
23. Yang, P., Sun, X., Li, W., Ma, S., Wu, W., Wang, H.: Sgm: sequence generation model for multi-label classification. arXiv preprint arXiv:1806.04822 (2018)
24. You, R., Dai, S., Zhang, Z., Mamitsuka, H., Zhu, S.: Attentionxml: Extreme multi-label text classification with multi-label attention based recurrent neural networks. arXiv preprint arXiv:1811.01727 (2018)
25. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Advances in neural information processing systems. pp. 649–657 (2015)
26. Zhou, C., Sun, C., Liu, Z., Lau, F.: A c-lstm neural network for text classification. arXiv preprint arXiv:1511.08630 (2015)
27. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)