Even the Simplest Baseline Needs Careful Re-investigation: A Case Study on XML-CNN

Si-An Chen[1,2], Jie-Jyun Liu[2], Tsung-Han Yang[2], Hsuan-Tien Lin[1], and Chih-Jen Lin[1]

[1]National Taiwan University
{d09922007,htlin,cjlin}@csie.ntu.edu.tw
[2]ASUS Intelligent Cloud Services
{eleven1_liu,henry1_yang}@asus.com

Abstract

The power and the potential of deep learning models attract many researchers to design advanced and sophisticated architectures. Nevertheless, the progress is sometimes unreal due to various possible reasons. In this work, through an astonishing example we argue that more efforts should be paid to ensure the progress in developing a new deep learning method. For a highly influential multi-label text classification method XML-CNN, we show that the superior performance claimed in the original paper was mainly due to some unbelievable coincidences. We re-examine XML-CNN and make a re-implementation which reveals some contradictory findings to the claims in the original paper. Our study suggests suitable baselines for multi-label text classification tasks and confirms that the progress on a new architecture cannot be confidently justified without a cautious investigation.

1 Introduction

Deep learning has been a popular research topic in NLP due to its superior performance. The intrinsic structure of deep learning allows researchers to enhance the model performance by introducing more complex network architectures. Nevertheless, the increasing complexity brings difficulties to ensure the true architectural progress. For example, Adhikari et al. (2019) have shown that LSTM architectures with appropriate regularization are either competitive or superior to more recent models. As another example, Liu et al. (2021) report that the lack of hyperparameter tuning in an influential work (Mullenbach et al., 2018) makes the progress of subsequent network developments questionable. Complex architectures are more difficult to train, involve more hyperparameters, and are riskier to unintentional implementation. Because new architectures are usually modified from previous ones, a questionable work may make the research progress unclear. Therefore, re-examining or reproducing influential architectures are now considered important in the community.

In this work, we re-examine XML-CNN (Liu et al., 2017), an influential work in extreme multi-label text classification (XMTC), as a case study to demonstrate the demands of inspecting existing architectures. XML-CNN has been viewed as an essential baseline in subsequent works (Peng et al., 2018; Prabhu et al., 2018; You et al., 2019; Chang et al., 2020; Adhikari et al., 2019) with more than hundreds of citations. XML-CNN roots from Kim-CNN (Kim, 2014), a classical architecture for multi-class text classification. The authors of XML-CNN proposed several modifications from Kim-CNN to accommodate the XMTC task and empirically claim that all modifications bring significant improvements.

Despite XML-CNN's popularity, we identified two serious implementation issues that make the original claims uncertain. First, the authors introduced dynamic max-pooling into XML-CNN, but the implementation is actually far from the intended formulation. Second, a bug in the experiment code caused the dimensions of convolution operations accidentally swapped. The two issues coincidentally make XML-CNN competitive, leading the authors to illusively claim superiority over Kim-CNN and usefulness of dynamic max-pooling in the original paper (Liu et al., 2017).

Our contribution can be summarized as follows.

- We point out, analyze, and correct the issues in the authors' XML-CNN implementation. Our implementation is made public to help the community build future works on top of the correct implementation
- We re-examine the claims about XML-CNN. Our results demonstrate that the progress from Kim-CNN to XML-CNN may not be as significant as claimed in Liu et al. (2017), and again confirm that careful attention is needed on ensuring true architectural progress.

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1987 - 2000

- Our investigation suggests that instead of XML-CNN, Kim-CNN or a simpler variant of XML-CNN should be considered as a baseline in XMTC tasks.

The paper is organized as follows: in Section 2, we introduce Kim-CNN, XML-CNN, and their differences. We conduct an investigation on XML-CNN in Section 3. The investigation includes inspection of the authors' code and our analysis on why it coincidentally works. We then conduct a fair and thorough comparison between Kim-CNN and XML-CNN in Section 4. Finally, we conclude this work in Section 5. Supplementary materials and programs used for experiments are available at https://www.csie.ntu.edu.tw/~cjlin/papers/xmlcnn/.

2 XML-CNN: CNN for Multi-Label Text Classification

For multi-label text classification, each instance is an n-word document that is associated with a subset of L possible categories. The relationship between the document and the categories can be modeled by a convolutional neural network (CNN), as pioneered for multi-class text classification by Kim-CNN (Kim, 2014). The architecture was later extended to XML-CNN (Liu et al., 2017) for multi-label text classification. Here we introduce the two architectures along with a focus on the key modifications.

2.1 CNN for Text Classification

Kim-CNN (Kim, 2014) is the first work that applies convolutional neural networks in text classification. The architecture is illustrated in Fig. 1a. Kim-CNN preprocesses a document by first encoding the i-th word to a k-dimensional embedding vector $\boldsymbol{x}_i \in \mathbb{R}^k$ (Pennington et al., 2014). We denote an n-word document by $\boldsymbol{x}_{1:n}$, where $\boldsymbol{x}_{i:j} = [\boldsymbol{x}_i, \dots, \boldsymbol{x}_j]^\top \in \mathbb{R}^{(j-i+1)\times k}$ represents a sub-sequence from the i-th to the j-th word in the document.

A convolutional operation applies a filter $\boldsymbol{w}_i \in \mathbb{R}^{m\times k}$ to a sub-sequence of m words to produce a new feature:

$$c_i = f(\boldsymbol{w}_i \cdot \boldsymbol{x}_{i:i+m-1} + b_i), \tag{1}$$

where f is an activation function such as ReLU, $b_i \in \mathbb{R}$ is a bias term and the "$\cdot$" operator means the sum after component-wise products between two matrices. The filter is applied to all m-word sub-sequences in the document to form a feature map $\boldsymbol{c} = [c_1, \dots, c_{n-m+1}] \in \mathbb{R}^{n-m+1}$. Suppose Kim-CNN uses t filters and let $\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(t)}$ be the corresponding feature maps. A max-pooling layer is then applied to summarize the features as $\boldsymbol{z} = \left[\max(\boldsymbol{c}^{(1)}), \dots, \max(\boldsymbol{c}^{(t)})\right] \in \mathbb{R}^t$. Lastly, a dropout layer and a fully-connected layer is used to predict a score vector

$$\boldsymbol{s} = \tilde{\boldsymbol{W}}(\boldsymbol{z} \odot \boldsymbol{r}) + \tilde{\boldsymbol{b}} \in \mathbb{R}^L, \tag{2}$$

where $\odot$ is the element-wise multiplication operator, $\tilde{\boldsymbol{W}} \in \mathbb{R}^{L\times t}, \tilde{\boldsymbol{b}} \in \mathbb{R}^L$ are learnable parameters and each r_i of $\boldsymbol{r} \in \mathbb{R}^t$ is a dropout random variable that follows a Bernoulli distribution.

Kim-CNN was originally proposed for multi-class classification based on the cross-entropy loss

$$-\sum_{i=1}^{L} y_i \log p_i, \quad \text{where } p_i = \frac{e^{s_i}}{\sum_{j=1}^{L} e^{s_j}} \tag{3}$$

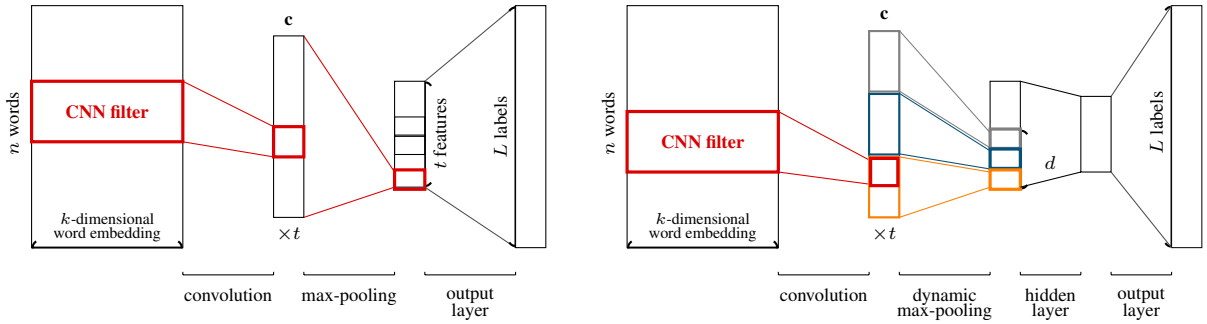
is the estimated probability of the i-th class, s_i is the i-th element of $\boldsymbol{s}$ that denotes the score of the i-th class, and $\boldsymbol{y} \in \{0,1\}^L$ denotes the ground truth of the instance. If the i-th label is associated with the document, then $y_i = 1$; otherwise, $y_i = 0$. By the construction of p_i in Eq. (3), $\sum_i p_i$ is forced to be 1, which is natural for multi-class classification. For multi-label classification, however, it is not clear whether requiring all p_i's to sum to one would be too restrictive, given that there can be multiple y_i's with $y_i = 1$. Nevertheless, the loss has been considered for some multi-label works (Gong et al., 2014; Ghosh et al., 2015).

2.2 From CNN to XML-CNN

XML-CNN is a pioneering work that extends Kim-CNN from multi-class text classification to XMTC. The architecture of XML-CNN is illustrated in Fig. 1b. It extends Kim-CNN with three modifications:

- using a label-wise binary cross-entropy loss instead of the cross-entropy loss in Eq. (3),
- adding an additional linear hidden layer with dropout,
- introducing dynamic max-pooling (Chen et al., 2015) to extract multiple features from each CNN filter.

For the first modification, the authors noticed the issue of the cross-entropy loss discussed in Section 2.1. They then allow the model to flexibly



(a) Kim-CNN (b) XML-CNN

Figure 1: Architectures of Kim-CNN and XML-CNN.

predict multiple positive labels by taking the independent binary cross-entropy loss instead:

$$-\sum_{i=1}^{L}\left[y_i \log(\sigma(s_i)) + (1-y_i)\log(1-\sigma(s_i))\right], \tag{4}$$

where $\sigma(s) = \frac{1}{1+e^{-s}}$ is the sigmoid function.

For the second modification, the additional linear layer may help to reduce the number of parameters, allowing the model to be stored in common GPU devices when L is extremely large. Let h be the number of elements in the added hidden layer. XML-CNN reduces the number of parameters after the CNN layer from $t \times L$ in the original Kim-CNN to

$$t \times h + h \times L \tag{5}$$

when h is sufficiently small.

For the third modification, the authors applied dynamic max-pooling (Chen et al., 2015) in XML-CNN to capture multiple features from different parts of the document. In contrast to the traditional max-pooling, which calculates the maximum along the whole sequence, dynamic max-pooling divides the sequence into multiple pools and then collects the maximum values within each pool to get some fine-grained features. Given a filter map $\boldsymbol{c} \in \mathbb{R}^n$,[1] the formulation with d pools is:

$$D(\boldsymbol{c}) = \left[\max\{\boldsymbol{c}_{1:\frac{n}{d}}\}, \ldots, \max\{\boldsymbol{c}_{n-\frac{n}{d}+1:n}\}\right] \in \mathbb{R}^d. \tag{6}$$

The output becomes

$$\boldsymbol{z} = \left[D(\boldsymbol{c}^{(1)}), \ldots, D(\boldsymbol{c}^{(t)})\right] \in \mathbb{R}^{dt}$$

instead of $\left[\max(\boldsymbol{c}^{(1)}), \ldots, \max(\boldsymbol{c}^{(t)})\right] \in \mathbb{R}^t$ in Kim-CNN.

2.3 Claims about XML-CNN

Liu et al. (2017) compared their proposed XML-CNN with Kim-CNN by reporting P@K on six datasets,[2] as shown in Table 1. P@K calculates for each document the percentage of correct predictions (i.e., precision) among the top K predicted labels and reports the average over all test documents. Table 1 clearly indicates significant improvements from Kim-CNN to XML-CNN on all datasets. To examine the impact of each new component in XML-CNN, the authors further conducted ablation studies to make the following claims.

- Eq. (4) is more suitable than Eq. (3) for multi-label classification problems.
- The additional linear layer improves both the performance and the scalability.
- Dynamic max-pooling further improves the performance significantly.

The impressive progress of XML-CNN makes it a standard benchmark for XMTC (e.g., Peng et al., 2018; Prabhu et al., 2018; You et al., 2019; Chang et al., 2020; Adhikari et al., 2019). However, we will show in this study that the progress may not be as significant as the authors claimed. While the first modification is included in our evaluation in Section 4.2, our focus is on the other two modifications, which correspond to the differences between XML-CNN and Kim-CNN-Eq.(4), the multi-label version equipped with the binary cross-entropy loss in Eq. (4). Subsequently, Kim-CNN-Eq.(4) will be shorthanded Kim-CNN for simplicity.

[1] Though we use $\boldsymbol{c} \in \mathbb{R}^{n-m+1}$ earlier, we let $\boldsymbol{c} \in \mathbb{R}^n$ here for easier explanation.

[2] Due to the space limit, we leave their and our NDCG@K results in the supplementary materials.

	RCV1			Amazon-670K		
	P@1	P@3	P@5	P@1	P@3	P@5
Kim-CNN-Eq.(3)	93.54	76.15	52.94	15.19	13.78	12.64
XML-CNN	96.86	81.11	56.07	35.39	31.93	29.32
	EUR-Lex			Wiki-30K		
	P@1	P@3	P@5	P@1	P@3	P@5
Kim-CNN-Eq.(3)	42.84	34.92	29.01	78.93	55.48	45.05
XML-CNN	76.38	62.81	51.41	84.06	73.96	64.11
	Amazon-12K			Wiki-500K		
	P@1	P@3	P@5	P@1	P@3	P@5
Kim-CNN-Eq.(3)	90.31	74.34	58.78	23.38	11.95	8.59
XML-CNN	95.06	79.86	63.91	59.85	39.28	29.81

Table 1: Results reported in Liu et al. (2017), where Kim-CNN-Eq.(3) indicates the setting to optimize Eq. (3) rather than Eq. (4).

3 Investigation into XML-CNN

In this section, we point out a significant gap between the formulations in the XML-CNN paper and the authors' implementations. We first reproduce the results in Liu et al. (2017) to ensure what the authors have done. We then confirm the reported superiority of XML-CNN over Kim-CNN is actually due to some coincidences.

3.1 The Challenges of Reproducing Liu et al. (2017)

The authors have released their implementation of XML-CNN on GitHub.[3] They wrote the code in Lasagne (Dieleman et al., 2015), an outdated deep learning framework. To facilitate a thorough comparison, we implement a PyTorch-based program[4] that is as close to the released Lasagne code as possible. Though their implementation is available, to our surprise, reproducing XML-CNN results on the same datasets is more challenging than expected. We leave details of solving various challenges in Appendix A. In particular, we find that some data sets used in Liu et al. (2017) are no longer available, so similar ones are considered; see data statistics in Table 2.

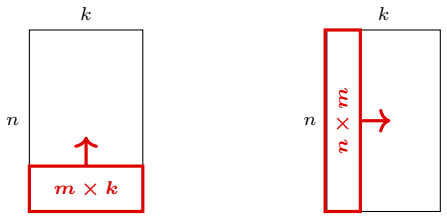
We choose EUR-Lex for checking the reproducibility due to the following reasons.

- The dataset is publicly available and from Tabel 2 it has the same statistics as in Liu et al. (2017).
- The improvement of XML-CNN is significant as shown in Table 1.
- The size is relatively small but adequate.

The results of the authors and our implementations

[3] https://github.com/jimmy646/XML-CNN

[4] The implementation is based on LibMultiLabel (released under the MIT license): https://github.com/ASUS-AICS/LibMultiLabel



(a) Normal CNN that follows Eq. (1) to go along the words.

(b) CNN in the public code of Liu et al. (2017) that goes along the embedding dimension.

Figure 2: Two implementations of CNN.

are respectively shown in the second and the third rows in Table 3. The difference between the two implementation is even smaller than the difference between XML-CNN's paper numbers to its public implementation. This justifies that our results are close enough for reproducing the numbers. We conclude that the author's result on EUR-Lex can be reproduced, though many issues must be addressed in the entire process.

3.2 Problematic Gap Between Implementations and Formulations

Though we can reproduce the results reported in Liu et al. (2017), in checking their programs, we surprisingly found some significant gaps between the implementation and the formulations in their paper. The first one is that their implementation of the convolutional operation is completely different from Eq. (1). We illustrate the two CNNs in Fig. 2. In the authors' implementation, the convolution operation sweeps along the embeddings rather than the words, as shown in Fig. 2b. That is, it seems the authors did not implement what they intended to do.

Another problem is about the dynamic max-

Dataset	# training data	# test data	# labels
EUR-Lex	15,449	3,865	3,956
Wiki-30K → Wiki10-31K	12,959 → 14,146	5,992 → 6,616	29,947 → 30,938
Amazon-12K → AmazonCat-13K	490,310 → 1,186,239	152,981 → 306,782	12,277 → 13,330
Amazon-670K	490,449	153,025	670,091
Wiki-500K → Wiki-500K	1,646,302 → 1,779,881	711,542 → 769,421	501,069 → 501,008

Table 2: The datasets used in Liu et al. (2017). Some are no longer available, so similar ones are considered in this work and "→" indicates the difference.

Implementation	Framework	P@1	P@3	P@5
Results reported in Liu et al. (2017)		76.38	62.81	51.41
Public code by Liu et al. (2017)'s authors	Lasagne	74.28	58.98	48.16
Our code mimicking the above	PyTorch	75.50	60.47	49.38

Table 3: Reproducing results reported in Liu et al. (2017) on EUR-Lex by using the authors' and our implementations.

pooling. The authors set the default pool size to 2 and stride to 1 in their public implementation:

$$[\max\{\boldsymbol{c}_{1:2}\}, \max\{\boldsymbol{c}_{2:3}\}, \dots \max\{\boldsymbol{c}_{n-1:n}\}], \quad (7)$$

which differs from Eq. (6) in that overlapped pools are used. Further, given that the aim of max-pooling is to extract information from each pool, a size-two pool is unusually small. We do not know why the authors implemented dynamic max-pooling in this form, but we will show that this odd implementation, together with the wrong convolution mentioned earlier, surprisingly works well.

To compare with these unusual settings, we generate another implementation of XML-CNN by following Eq. (1) and Eq. (6). The details of our implementation can be found in Appendix B. According to Liu et al. (2017), this version should be what its authors intend to have. Table 4 shows the results on EUR-Lex by various ways to implement Kim-CNN and XML-CNN. Other settings (e.g., hyperparameters) are kept to be the same as those in the authors' implementation; see also Section 3.1 and supplementary materials.[5] From Table 4, we have the following observations.

- For each category (Kim-CNN or XML-CNN), the last row indicates the setting described in the original papers. If the CNN input is changed to the wrong one, the results of both Kim-CNN and XML-CNN become dramatically worse (rows 1 and 5). On the other hand, if the implementation of dynamic max-pooling follows Eq. (7) rather than Eq. (6), the result of XML-CNN also significantly deteriorates (row 4).
- However, if both inappropriate settings for CNN and dynamic max-pooling are applied, the resulting procedure corresponds to the actual implementation by Liu et al. and works surprisingly well (row 3). In contrast, without the help of Eq. (7), Kim-CNN by the inappropriate CNN implementation performs poorly. In such a situation, Kim-CNN's scores are quite similar to the results of Kim-CNN reported in Liu et al. (2017), as shown in Table 1. So we presume that in Liu et al. (2017), Kim-CNN was implemented with the inappropriate CNN setting.

In sum, the implementation seems not what Liu et al. intended to do in their paper. Thus, their conclusions based on the unintentional implementations may be questionable. In particular, in Table 4 Kim-CNN is competitive if an implementation following its original paper (Kim, 2014) is considered.

For better distinction in subsequent discussions, we name the two XML-CNN implementations respectively corresponding to Liu et al.'s paper and public code as follows.

- **XML-CNN-paper**: XML-CNN following Eq. (1) and Eq. (6).
- **XML-CNN-impl**: XML-CNN using CNNs sweeping along embeddings and Eq. (7).

3.3 The Two Implementations of XML-CNN: Analysis

We try to explain why XML-CNN-impl can achieve competitive results. For the analysis, we first argue that conceptually, the unusual dynamic max-pooling Eq. (7) is similar to not doing pooling. The reason is because the small pool size $= 2$ implies that at least half of $\boldsymbol{c}$ elements are retained. Then we design an experiment to compare the combina-

[5] As mentioned in Section 2.3, we note again that Kim-CNN in all our experiments, unless specified, optimizes the same loss in Eq. (4) as XML-CNN.

Method	CNN sweeping direction	Dynamic Max-pooling	P@1	P@3	P@5	Note
Kim-CNN	embeddings	N/A	45.38	34.02	27.72	Actual implementation of Liu et al. (2017)
Kim-CNN	words	N/A	75.83	61.08	50.19	Procedure described in Kim (2014)
XML-CNN	embeddings	Eq. (7)	75.96	60.56	49.23	XML-CNN-impl: actual implementation of Liu et al. (2017)
XML-CNN	words	Eq. (7)	58.09	45.19	37.06	
XML-CNN	embeddings	Eq. (6)	63.03	48.31	39.32	
XML-CNN	words	Eq. (6)	75.73	61.82	50.82	XML-CNN-paper: procedure described in Liu et al. (2017)

Table 4: Results of different implementations of the convolutional layer and dynamic max-pooling for Kim-CNN and XML-CNN. The standard CNN input should be $k \times n$. Other settings are the same as those for the last two rows of Table 3.

tions of the following settings.

- CNN sweeping direction: embeddings or words
- Pooling implementation: standard max-pooing or no pooling

Table 5 shows the P@1 scores on predicting the test set of EUR-Lex. We have the following observations.

- Results of the (embeddings, no pooling) setting are similar to those of the (embeddings, Eq. (7)) setting in Table 4. This confirms our earlier argument that Eq. (7) is close to no pooling.
- If CNN sweeps along the words, the standard max-pooling is significantly better than no pooling. A possible explanation is that when CNN sweeps along the words, some sub-sequence of words are shown to be more important than others. Then the standard max-pooling is helpful to identify them. This situation is similar to that in image classification, where max-pooling is effective to extract "sharp" features (Springenberg et al., 2015).
- If CNN sweeps along the embeddings, an opposite situation occurs. No pooling is much better than standard max-pooling. Because all the embeddings can be considered equally useful, the resulting features after convolutional operation have similar importance. For such "smooth" features, it is known in image classification that average pooling or no pooling are recommend (Springenberg et al., 2015). In other words, standard max-pooling can extract little information in such a case and may lead to worse performance.

In Section 3.4, we will present results to further support the above analysis.

	$\max\{\boldsymbol{c}\}$	No pooling
words	74.67	53.61
embeddings	58.14	76.48

Table 5: P@1 of combinations of CNN sweeping directions and pooling methods for implementing XML-CNN. Note that the first column differs from the first two rows in Table 4 because we now have a hidden layer.

3.4 The Two Implementations of XML-CNN: Performance Comparison

Table 6 shows a comprehensive comparison between XML-CNN-impl and XML-CNN-paper on more datasets. In contrast to Table 4 where we follow the hyperparameters used in Liu et al. (2017), we tune the hyperparameters for both methods in Table 6.[6] We observe that XML-CNN-paper outperforms XML-CNN-impl on EUR-Lex and Wiki10-31K. Following the discussion in Section 3.3, the reason may be that XML-CNN-impl lacks the ability to learn position-agnostic features when documents are long. Note that for EUR-Lex and Wiki10-31K, the documents are truncated to 500 words because of the long document length. On the other hand, XML-CNN-impl works competitively on AmazonCat-13K and Amazon-670K. Though the documents are also truncated to 500 words when needed, the average document lengths of these two sets are less than 250.

In sum, XML-CNN-paper should be preferable

[6]Details of experimental settings and hyperparameter search are in Section 4.1 and Appendix C.

because of the more reasonable architecture and better performance on long documents. Moreover, XML-CNN can deal with variable sentence lengths, while XML-CNN-impl cannot because the network architecture depends on the sentence length. We consider XML-CNN-paper as the setting for XML-CNN in subsequent experiments.

4 The True Performance of XML-CNN

After showing the gap between the implementation and the formulation in Liu et al. (2017), the true performance of XML-CNN should be re-examined. In this section, we conduct a comprehensive ablation study for XML-CNN to investigate the claims in the original paper. We then investigate more deeply on dynamic max-pooling to determine its usefulness for XMTC tasks. The results bring us a similarly competitive but simpler baseline for XMTC tasks.

4.1 Experimental Setup

We consider a random 80/20 split of the training data to generate a training subset and a validation subset for hyperparameter selection. We follow Liu et al. (2017) to truncate the documents to 500 words, represent each word as a 300-dimensional GloVe word embedding (Pennington et al., 2014) and pad the sequences in each batch when needed. The word embeddings are considered as trainable parameters during training. We have carefully conducted hyperparameter selection. The procedure and other details are in Appendix C.

We follow Liu et al. (2017) to train the models with 50 epochs on the whole training set after hyperparameter tuning and then evaluate the test set. We evaluate each method on three datasets: EUR-Lex, Wiki10-31K, and AmazonCat-13K. In Section 4.2, we conduct the ablation study by including one larger dataset: Amazon-670K. All of them are in English. The datasets are obtained from the repository of You et al. (2019) and we follow Liu et al. (2017) to reduce the vocabulary set.[7]

4.2 Ablation Studies of XML-CNN

To understand how each component introduced in Liu et al. (2017) really works, we conduct an ablation study as what the authors have done. Specifically, the effects of using Eq. (4) as the loss, adding a linear hidden layer, and introducing dynamic max-pooling are checked. By the results shown in Table 7, we can re-examine what the authors have claimed in Liu et al. (2017). To begin, from the first and the second rows, the loss function Eq. (4) indeed improves the scores 2%-6% on each dataset. The results validate the claim that Eq. (4) is more suitable for multi-label tasks than Eq. (3).

Next, while adding a hidden layer is claimed to be beneficial in Liu et al. (2017), our results show that the hidden layer is slightly harmful on EUR-Lex and Wiki10-31K when the standard max-pooling is applied; see rows 2 and 4 in Table 7. It works when dynamic max-pooling is employed, but the improvements are not significant. The authors also claimed that the hidden layer could reduce the number of parameters; see Eq. (5). This claim is true when h is relatively small compared with the number of convolutional features. However, we noticed that larger h's such as 512 and $1,024$ are always preferable after hyperparameter tuning. In these cases, the number of parameters may not be reduced.

Lastly, we discuss the effect of dynamic max-pooling. In the situation of not adding a hidden layer, dynamic max-pooling slightly improves upon the standard max-pooling on most but not all datasets. If a hidden layer is included in the architecture, dynamic max-pooling also gives moderate improvements. However, dynamic max-pooling may require more network parameters due to multiple pools. To check its usefulness, we investigate more in Section 4.3.

4.3 Further Investigation on Dynamic Max-Pooling

We conduct two experiments to understand whether dynamic max-pooling always benefits XML-CNN. The first experiment is a comparison between different numbers of pools. The second experiment compares dynamic max-pooling with standard max-pooling by using a similar total number of parameters.[8] The experiment results and more discussions are in Appendix D.1 and Appendix D.2. The investigations tell us:

- Using too many pools may deteriorate the performance.
- Under similar total numbers of parameters, standard max pooling is more preferable than dynamic max-pooling.

[7] See details in Appendix A.4.

[8] For dynamic max-pooling, the output size $t \times d$ is proportional to the pool size d. For standard max-pooling ($d = 1$), we can enlarge the number of filters t to have a similar number of parameters; see details in Appendix D.2.

method	EUR-Lex			Wiki10-31K		
	P@1	P@3	P@5	P@1	P@3	P@5
XML-CNN-impl	77.39	62.32	51.28	83.98	70.03	60.18
XML-CNN-paper	78.94	65.77	54.15	84.70	71.80	61.03
	AmazonCat-13K			Amazon-670K		
	P@1	P@3	P@5	P@1	P@3	P@5
XML-CNN-impl	94.73	80.29	64.97	38.02	34.13	31.20
XML-CNN-paper	94.78	80.03	64.52	35.69	31.89	29.08

Table 6: Comparison between the two XML-CNN implementations. XML-CNN-impl is the actual implementation of Liu et al. (2017). XML-CNN-paper is our implementation that follows Liu et al. (2017).

loss function	hidden layer	max-pooling	EUR-Lex			Wiki10-31K			Note
			P@1	P@3	P@5	P@1	P@3	P@5	
Eq. (3)	N	standard	72.78	59.84	49.94	80.70	64.83	55.43	Kim-CNN (Kim, 2014)
Eq. (4)	N	standard	80.93	66.38	55.34	82.78	68.07	57.63	
Eq. (4)	N	dynamic	77.88	64.58	53.38	83.37	70.64	60.16	
Eq. (4)	Y	standard	76.56	62.92	51.84	81.73	68.82	58.65	
Eq. (4)	Y	dynamic	78.94	65.77	54.15	84.70	71.80	61.03	XML-CNN (Liu et al., 2017)
			AmazonCat-13K			Amazon-670K			
			P@1	P@3	P@5	P@1	P@3	P@5	
Eq. (3)	N	standard	92.85	76.90	61.76	27.23	24.65	22.70	Kim-CNN (Kim, 2014)
Eq. (4)	N	standard	93.41	78.11	62.95	33.38	29.99	27.47	
Eq. (4)	N	dynamic	93.65	78.56	63.41	34.61	30.91	28.25	
Eq. (4)	Y	standard	94.73	79.64	63.94	33.86	30.27	27.69	
Eq. (4)	Y	dynamic	94.78	80.03	64.52	35.69	31.89	29.08	XML-CNN (Liu et al., 2017)

Table 7: An ablation study of XML-CNN. For max-pooling, "standard" means the standard way of using the single maximal value, while "dynamic" means to use Eq. (6).

4.4 What We May Claim about XML-CNN

We conclude our findings in this section as follows:

- Eq. (4) is indeed more suitable for multi-label tasks than Eq. (3).
- For the hidden layer, there is a minor tradeoff between the number of parameters and the performance. A negative way to interpret this is that introducing the hidden layer does not always improve the performance. However, a positive interpretation is that with a slight performance loss, a hidden layer can effective reduce the number of parameters when the output size of the pooling operation is large.
- Dynamic max-pooling is not as beneficial as increasing the number of convolutional filters.

After our careful re-investigation, our suggestion to future studies of XMTC is that instead of using XML-CNN as a baseline, the following simpler settings can be considered.

- If there is no memory concern, Kim-CNN is suitable for its similar performance to XML-CNN.
- Otherwise, a simplified version of XML-CNN without dynamic max-pooling, namely Kim-CNN with an additional hidden layer, is sufficiently strong and space-efficient as the baseline.

5 Conclusion

This work aims to highlight the importance of validating existing works. From the investigation of XML-CNN, we learned that there are many pitfalls when developing new architectures. We correct the issues in the authors' implementation, carefully re-examine the claims about XML-CNN and recommend suitable baselines for future studies. Though not proposing a new method, we hope this work encourages the community to reproduce and re-examine influential works. This may help the community build future works on top of correct materials.

6 Acknowledgement

This work is partially supported by ASUS Intelligence Cloud Services and the Ministry of Science and Technology of Taiwan via the grants MOST 110-2628-E-002-013. We also thank the National Center for High-performance Computing (NCHC) of National Applied Research Laborato-

ries (NARLabs) in Taiwan for providing computational resources.

References

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. Rethinking complex neural network architectures for document classification. In *NAACL-HLT*.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *KDD*.

Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. 2020. Taming pretrained transformers for extreme multi-label text classification. In *KDD*.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL*.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, et al. 2015. Lasagne: First release.

Sayan Ghosh, Eugene Laksana, Stefan Scherer, and Louis-Philippe Morency. 2015. A multi-label convolutional neural network approach to cross-domain action unit detection. In *ACII*.

Yunchao Gong, Yangqing Jia, Thomas Leung, Alexander Toshev, and Sergey Ioffe. 2014. Deep convolutional ranking for multilabel image annotation. In *ICLR*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Jie-Jyun Liu, Tsung-Han Yang, Si-An Chen, and Chih-Jen Lin. 2021. Parameter selection: Why we should pay more attention to it. In *ACL*.

Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *SIGIR*.

James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. 2018. Explainable prediction of medical codes from clinical text. In *NAACL*.

Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-CNN. In *WWW*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

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 *WWW*.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. 2015. Striving for simplicity: The all convolutional net. In *ICLR Workshop*.

Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2019. AttentionXML: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *NeurIPS*.

A The Challenges of Reproducing Liu et al. (2017)

A.1 Dataset

The authors evaluated XML-CNN on six datasets from the Extreme Multi-Label Repository.[9] Unfortunately, some of the datasets have been changed on the repository to different number of data/labels (with a similar name). Furthermore, for some of them, the repository does not provide the raw-text documents, making it hard to preprocess the documents to the embedding needed for XML-CNN. Fortunately, we find the repository of AttentionXML.[10] (You et al., 2019) where two datasets (EUR-Lex and Amazon-670K) of raw text match the statistics of the datasets in the XML-CNN work. We then choose the smaller EUR-Lex as the first attempt to reproduce XML-CNN faithfully.

A.2 Evaluation

The released code includes only the training but not the validation/evaluation procedure. In the original paper (Liu et al., 2017), it is mentioned that 25% of training data is reserved as the validation set for hyperparameter selection. However, the details such as how to generate the validation set, which metric was considered in validation, and whether they re-trained the model with the whole training set are not specified in the paper. Therefore, we cannot exactly replicate the results in Table 1. We ran the released code and observed that with only 75% of training data, the results are always worse than ones reported in Liu et al. (2017). Thus, we presume that in Liu et al. (2017), the authors reported the results of models trained on the whole training set by using the selected hyperparameters.

A.3 Lasagne vs PyTorch

As mentioned in Sec 3.1, we implement a PyTorch-based program that is as close to the released Lasagne code as possible. We fix the common hyperparameters such as the number of filters and the dropout rate as ones provided in the authors' implementation. Then we train the whole training set and follow their setting to report the test scores at the 50-th epoch. The results of their and our implementations are respectively shown in the second and the third rows in Table 8. The minor differences between the scores are possible because ensuring everything to be the same from the beginning to the end is tremendously difficult. For example, optimizers implemented in Lasagne and PyTorch are not entirely the same. What we have confirmed is that for the network part, under the same input, the two implementations generate exactly the same output and loss values. Therefore, we conclude that our implementation can be used together with theirs in checking the reproducibility. However, both are still worse than the results of Liu et al. (2017) in the first row of Table 8. This fact encouraged us to investigate more on the data processing step done in Liu et al. (2017).

A.4 Vocabulary Set

In Liu et al. (2017), the authors compared XML-CNN with some linear-based algorithms, which use bag-of-word (BOW) features to deal with document inputs. The BOW features usually only consider vocabularies with higher frequency to reduce the dimensionality. To fairly compare XML-CNN with linear models, the authors removed vocabularies which are not used in the BOW features. While the Extreme Multi-Label Repository provides BOW features of EUR-Lex and we assume that they were used in Liu et al. (2017), the vocabulary set of the BOW features is not accessible now. Fortunately, we obtained from the repository owner the vocabulary set used in generating their BOW features. The results of XML-CNN with the reduced vocabulary set are shown in the fourth and the fifth rows in Table 8. By using the reduced vocabulary set, the results of both Lasagne and PyTorch implementations are improved to be closer to the ones in Table 1. As a result, we conclude that the authors' result on EUR-Lex can be reproduced, though many issues must be addressed in the entire process.

B Implementation Details of Section 3.2

To implement Eq. (6), we follow Adhikari et al. (2019) to use AdaptiveMaxPool1d[11] and consider 4 pools, i.e., $d = 4$. Notice that Eq. (6) does not handle the situation where the sequence length is not divisible by d. Adaptive max-pooling solve the problem by allowing some overlapping between

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

[10] The repository is free for non-commercial use. https://github.com/yourh/AttentionXML

[11] https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveMaxPool1d.html

Implementation	Framework	Vocabulary set	P@1	P@3	P@5
Results reported in Liu et al. (2017)			76.38	62.81	51.41
Public code by Liu et al. (2017)'s authors	Lasagne	All	72.08	56.32	46.20
Our code mimicking the above	PyTorch	All	74.05	59.61	48.24
Public code by Liu et al. (2017)'s authors	Lasagne	Reduced	74.28	58.98	48.16
Our code mimicking the above	PyTorch	Reduced	75.50	60.47	49.38

Table 8: Reproducing results reported in Liu et al. (2017) on EUR-Lex by using the authors' and our implementations.

parameter	range
learning rate	$\{0.0001, 0.0003, 0.001, 0.003, 0.01\}$
# of filters	$\{96, 192, 384, 768\}$
embedding dropout	$\{0.2, 0.4, 0.6, 0.8\}$
# pools	$\{2, 8\}$
hidden layer	$\{256, 512, 1024\}$
hidden layer dropout	$\{0.2, 0.4, 0.6, 0.8\}$

Table 9: The range of hyperparameters used for selection.

pools. Therefore, it generates exactly d outputs from documents with varying lengths.

C Details of Experimental Setup

From the hyperparameter ranges listed in Table 9, we apply Optuna (Akiba et al., 2019) to select the best hyperparameters from 48 random trials. In the validation procedure, we optimize P@1 for EUR-Lex, AmazonCat-13K, and Amazon-670K, and P@3 for Wiki10-31K. We do not optimize P@1 for Wiki10-31K because there is a dominant class that is associated with about 80% of data. Each trial is stopped if the validation metric does not improve for 10 epochs or when it reaches 50 epochs.

In the original papers of Kim-CNN and XML-CNN, both described the use of filters with different window sizes in the convolutional layer. In Kim (2014) and Liu et al. (2017), filter sizes $3, 4, 5$ and $2, 4, 8$ are respectively used. However, as shown in Table 10, using multiple filter sizes does not have a significant benefit compared with using a fixed filter size 8. Furthermore, among single filter-size settings, the filter size 8 is generally competitive, so we use it in our ensuing investigation.

The experiments are conducted on Azure with an Nvidia Tesla V100 GPU, taking <1, <1, 6, 20 GPU hours for one trial on EUR-Lex, Wiki10-31K, AmazonCat-13K, and Amazon-670K respectively.

D Further Investigation on Dynamic MaxPooling

D.1 Effect of the Number of Pools

In dynamic max-pooling, a crucial hyperparameter is the number of pools d. Nevertheless, in the public code of the XML-CNN work (Liu et al., 2017), due to the unusual setting in Eq. (7), d is not a fixed number but depends on the document length. Consequently, discussion about the number of pools is lacking in the original work.

In Table 11, we conduct a comparison by using $d \in \{1, 2, 8, 32, 64\}$. On all datasets, $d = 2$ and $d = 8$ have the best performance. Increasing the number of pools to more than 8 not only leads to worse results on some problems, but also costs more memory and training time. Our results indicate that while the goal of dynamic max-pooling is to extract multiple features from each CNN filter, using too many pools may deteriorate the performance instead.

D.2 Investigation on Dynamic Max-Pooling by Fixing the Number of Parameters

As discussed in Sec 4.2, we noticed that the number of parameters in XML-CNN increases along with the number of pools in dynamic max-pooling. Assume the number of filters is t and the number of pools is d. In XML-CNN, the total number of filters after the pooling layer is $t \times d$, while Kim-CNN still only has t filters. It is unclear whether the improvement of dynamic max-pooling is caused by the richer information from multiple pools or simply the larger number of parameters. We investigate this issue by comparing XML-CNN with different numbers of pools but ensuring the similar number of parameters. From Table 12, we observe that XML-CNN with 1 pool (i.e., without dynamic max-pooling) outperforms XML-CNN with 2 or 8 pools. The result reveals that the architectural modification of dynamic max-pooling may not be that useful.

Though increasing the number of filter also introduced more parameters in CNN, the number is negligible compared to the number of parameters in

method	filter sizes	EUR-Lex			Wiki10-31K			AmazonCat-13K		
		P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
Kim-CNN	[2]	78.45	63.41	52.46	83.30	69.34	59.17	93.48	78.21	63.03
	[4]	79.04	64.23	52.66	83.19	68.78	58.26	93.66	78.56	63.35
	[8]	80.23	66.12	54.48	83.07	69.98	59.96	93.75	78.97	63.93
	[2, 4, 8]	79.90	66.43	54.86	82.65	68.05	56.93	93.51	78.14	62.93
XML-CNN	[2]	75.83	62.35	51.67	82.91	69.20	58.97	94.59	79.80	64.32
	[4]	77.70	62.97	52.27	82.84	70.20	59.73	94.87	80.27	64.79
	[8]	77.98	65.11	53.90	82.68	69.40	59.43	94.55	79.72	64.16
	[2, 4, 8]	78.37	64.78	53.61	80.47	68.23	58.24	94.89	80.02	64.25

Table 10: P@K results of comparison between fixed filter size and multiple filter sizes.

# pools	EUR-Lex			Wiki10-31K			AmazonCat-13K		
	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
$d = 1$	76.40	62.78	51.88	80.89	67.89	58.17	94.73	79.64	63.95
$d = 2$	77.98	65.11	53.90	82.68	69.40	59.43	94.55	79.72	64.16
$d = 8$	76.79	63.28	52.10	84.19	71.55	61.14	94.79	80.04	64.49
$d = 32$	66.57	52.51	42.56	82.94	69.20	59.20	94.46	79.45	63.78
$d = 64$	68.28	54.14	44.19	83.01	69.91	59.13	94.29	79.00	63.27

Table 11: Effect of number of pools in dynamic max-pooling

the output layer, where the output size L is usually extremely large.

E NDCG results

This section shows NDCG@K results reported by Liu et al. (2017) and in our experiments. Table 13 shows NDCG@K results reported by Liu et al. (2017). Table 14 shows NDCG@K results corresponding to Table 4. Table 15 shows NDCG@K results of our ablation study (Table 7). The observations from NDCG@K results are similar to those from P@K results.

# of filters	# of pools	EUR-Lex			Wiki10-31K			AmazonCat-13K		
		P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
256	1	76.66	63.70	52.64	83.07	69.35	58.89	94.52	79.66	64.23
128	2	77.36	62.93	51.79	83.62	68.92	58.35	94.34	79.51	64.14
1024	1	78.37	65.65	54.42	83.42	70.66	60.50	94.90	80.29	64.79
128	8	75.91	62.12	51.33	84.11	69.94	59.22	94.30	79.50	64.13

Table 12: A comparison between different settings of XML-CNN with the same number of parameters.

	RCV1			Amazon-670K		
	N@1	N@3	N@5	N@1	N@3	N@5
Kim-CNN-Eq.(3)	93.54	88.2	87.26	15.19	14.6	14.12
XML-CNN	96.88	92.63	92.22	35.39	33.74	32.64
	EUR-Lex			Wiki-30K		
	N@1	N@3	N@5	N@1	N@3	N@5
Kim-CNN-Eq.(3)	42.84	36.95	33.83	78.93	60.52	51.96
XML-CNN	76.38	66.28	60.32	84.06	76.35	68.94
	Amazon-12K			Wiki-500K		
	N@1	N@3	N@5	N@1	N@3	N@5
Kim-CNN-Eq.(3)	90.31	83.87	81.21	23.38	15.45	13.64
XML-CNN	95.06	89.48	87.06	59.85	48.67	46.12

Table 13: NDCG@K results reported in Liu et al. (2017), where Kim-CNN-Eq.(3) indicates the setting to optimize Eq. (3) rather than Eq. (4).

Method	CNN sweeping direction	Dynamic Max-pooling	N@1	N@3	N@5	Note
Kim-CNN	embeddings	N/A	45.38	36.72	33.04	Actual implementation of Liu et al. (2017)
Kim-CNN	words	N/A	75.83	64.75	58.93	Procedure described in Kim (2014)
XML-CNN	embeddings	Eq. (7)	75.96	64.31	58.20	XML-CNN-impl: actual implementation of Liu et al. (2017)
XML-CNN	words	Eq. (7)	58.09	48.30	43.81	
XML-CNN	embeddings	Eq. (6)	63.03	51.92	46.88	
XML-CNN	words	Eq. (6)	75.73	65.31	59.54	XML-CNN-paper: procedure described in Liu et al. (2017)

Table 14: NDCG results of different implementations of the convolutional layer and dynamic max-pooling for Kim-CNN and XML-CNN.

loss function	hidden layer	max-pooling	EUR-Lex			Wiki10-31K			Note
			N@1	N@3	N@5	N@1	N@3	N@5	
Eq. (3)	N	standard	72.78	63.12	58.03	80.70	68.40	60.95	Kim-CNN (Kim, 2014)
Eq. (4)	N	standard	80.93	70.07	64.42	82.78	71.48	63.39	
Eq. (4)	N	dynamic	77.88	67.93	62.13	83.37	73.61	65.61	
Eq. (4)	Y	standard	76.56	66.40	60.60	81.73	71.81	64.01	
Eq. (4)	Y	dynamic	78.94	69.11	63.09	84.70	74.84	66.61	XML-CNN (Liu et al., 2017)
			AmazonCat-13K			Amazon-670K			
			N@1	N@3	N@5	N@1	N@3	N@5	
Eq. (3)	N	standard	92.85	85.90	83.50	27.23	26.02	25.20	Kim-CNN (Kim, 2014)
Eq. (4)	N	standard	93.41	87.03	84.77	33.38	31.70	30.60	
Eq. (4)	N	dynamic	93.65	87.41	85.22	34.61	32.69	31.49	
Eq. (4)	Y	standard	94.73	88.68	86.20	33.86	32.03	30.88	
Eq. (4)	Y	dynamic	94.78	88.99	86.72	35.69	33.72	32.45	XML-CNN (Liu et al., 2017)

Table 15: An ablation study of XML-CNN evaluated by NDCG@K. For max-pooling, "standard" means the standard way of using the single maximal value, while "dynamic" means to use Eq. (6).