Embedding Convolutions for Short Text Extreme Classification with Millions of Labels

Anonymous ACL submission

Abstract

Automatic annotation of short-text data to a large number of target labels, referred to as Short Text Extreme Classification, has found numerous applications including prediction of related searches and product recommendation tasks. In this paper, we propose a convolutional architecture INCEPTIONXML which is light-weight, yet powerful, and robust to the inherent lack of word-order in short-text queries encountered in search and recommendation tasks. We demonstrate the efficacy of applying convolutions by recasting the operation along the embedding dimension instead of the word dimension as done in conventional usage of CNNs for text classification. Towards scaling our model to problems with millions of labels, we also propose INCEPTIONXML+ framework. This addresses the shortcomings of the dynamic hard-negative mining framework in the recently proposed LIGHTXML by improving the alignment between the label-shortlister and extreme classifier. INCEPTIONXML+ is not only smaller than state-of-the-art deep extreme classifier, ASTEC, in terms of model size but also significantly outperforms it on popular benchmark datasets. For reproducibility, the code is made available with this submission.

1 Introduction

Extreme Multi-label Classification (**XML**) involves classifying instances into a set of most relevant labels from an extremely large (on the order of millions) set of all possible labels. For scenarios when the input instances are short text queries, many successful applications of the XML framework have been found in ranking and recommendation tasks such as prediction of *Related Search* on search engines (Jain et al., 2019), suggestion of query phrases corresponding to short textual description of products on e-stores (Chang et al., 2020) and product-to-product recommendation (Dahiya et al., 2021a; Chang et al., 2021).

Technical difficulties in XML: The XML task has two primary challenges: (i) extremely large number of possible labels which leads to both memory and computational bottleneck, and (ii) a large fraction of these being tail labels paired with a handful of positive samples (Jain et al., 2016) which makes it hard for the extreme classifier to learn rich representations for these tail labels.

Additional challenges from short-text queries: The above problems are exacerbated further as : (i) unlike documents, most short text queries are sparse and contain very few words and (ii) are typically plagued with noise and non-standard phrases which do not always observe the syntax of a written language (Tayal et al., 2020). Therefore, short texts give rise to a significant amount of ambiguity, making it hard to learn meaningful representations (Wang and Wang, 2016).

InceptionXML: To address the above, in this paper, we (i) develop INCEPTIONXML, a light-weight CNN-based encoder, which goes against the traditional paradigm (Kim, 2014; Liu et al., 2017) of convolving over the words dimension in favor of the embedding dimension, (ii) propose an embedding-enhancement module that projects the input to a word-order agnostic representation making our approach more robust to lack of structure in short-text queries, (iii) develop INCEPTIONXML+ to overcome the shortcomings of the dynamic hard-negative mining framework proposed in LIGHTXML (Jiang et al., 2021) by bringing the label-shortlisting task closer to the extreme classification task and scale our model to millions of labels, (iv) further the state-of-the-art on popular benchmarks by an average of 5% and 8% on P@K and propensity-scored P@K metrics respectively.

By balancing model complexity against computational complexity, INCEPTIONXML+ finds the sweet-spot between the two extreme ends of modern deep extreme classification pipelines. On one end exist the pre-trained transformer-based en-

coders (Jiang et al., 2021; Ye et al., 2020; Chang et al., 2020) that treat XML as a down-stream task, and as a result become unscalable to millions of labels due to their compute requirements. On the other end, are those which are built on under-fitting frugal architectures such as ASTEC (Dahiya et al., 2021b), and hence sacrifice accuracy to achieve scalability to millions of labels.

As the real-world use cases of extreme classification require very fast inference times and low memory loads, the deployment of large transformer-based architectures as in LIGHTXML (Jiang et al., 2021) leads to slower training and inference, making them an overkill for the task at hand. We show that replacing the transformer encoder with our lightweight CNN-based encoder, combined with further improvements to dynamic hard-negative mining technique leads to better prediction performance apart from faster training and the ability to scale to millions of labels. Also, by employing a much richer architecture in comparison to the frugal ASTEC model, we show that a single INCEPTIONXML model can even outperform an ensemble of three ASTEC learners on publicly available datasets by as much as 10% on P@k metrics.

2 Related Work

Extreme Classification: The focus of a majority of initial works in this domain has been on designing one-vs-rest (Babbar and Schölkopf, 2017), tree-based (Jain et al., 2016; Prabhu et al., 2018; Chalkidis et al., 2019; Khandagale et al., 2020) or label embedding based (Bhatia et al., 2015) classifiers with fixed features in the form of bag-of-words representation.

With advances in deep learning techniques which jointly learn the suitable feature representation and the final classifier, recent techniques based on attention mechanism (You et al., 2019) and pre-trained transformer models (Chang et al., 2020; Ye et al., 2020; Jiang et al., 2021; Yu et al., 2020) have recently shown great promise. Previous advances in the domain using CNNs have also been made. While (Wang et al., 2017) extended (Kim, 2014) for short text classification, (Liu et al., 2017) built upon the same for extreme classification.

Short-text Extreme Classification: In tasks where the inputs are short text queries, there has been a slew of works lately on designing deep extreme classifiers specialized to solve the problem. These include ASTEC (Dahiya et al., 2021b), DECAF (Mittal et al., 2021a), GALAXC, ECLARE (Mittal et al., 2021b), and SIAMESEXML (Dahiya et al., 2021a). Based on the availability of label meta-data, these can be divided into two categories: (i) ones which make no assumptions regarding label text, i.e., labels are numeric identifiers, and (ii) others which assume that the labels are endowed with clean label text. While ASTEC and INCEPTIONXML belong to the first category, the rest of the above mentioned algorithms identify with the second. Even though the additional label meta-data is useful, it is known for only a small subset of all labels. For instance, on AmazonTitles-3M, label text is available only for 1.3 million labels. On the other hand, the former problem setup, which is the focus of this work, makes no assumption about label-text, and hence is a harder, more general and widely applicable problem.

Drawbacks of conventional CNNs in short-text classification: Traditionally, in the usage of CNNs over words in text classification, the intent is to capture the occurrences of n-grams for representation learning (Kim, 2014; Liu et al., 2017). We argue that this formulation is unsuitable for short-text classification problems due to (i) the implicit but incorrect assumption of proper *word-ordering* in short-text queries (Wang and Wang, 2016), and (ii) as explained next, the much *smaller sequence length* that restricts the effectiveness of convolution in CNNs over the inputs.

In the datasets derived from Wikipedia titles, 98% documents have 8 or less words, while 82% have 4 words or less (Table: 4 in Appendix). Moreover, 70% of the instances in AmazonTitles-670K consist of 8 words or less (Figure: 5). This makes the convolutional filters spanning over 4-8 words in Kim (2014); Liu et al. (2017); Wang et al. (2017) behave analogously to a weak fully connected layer with very few hidden units, and hence leading to feature maps with very few activations which are sub-optimal for representation learning.

3 Embedding Convolutions

On looking closely at short-text queries, one can infer that the presence of a word in a query is more important than its exact position for extreme classification tasks. For example, the queries "*best wireless headphones 2021*" and "*2021 best headphones wireless*" (both of which are common in real-world scenarios) should result in similar search results on an e-commerce website (Tayal et al., 2020).

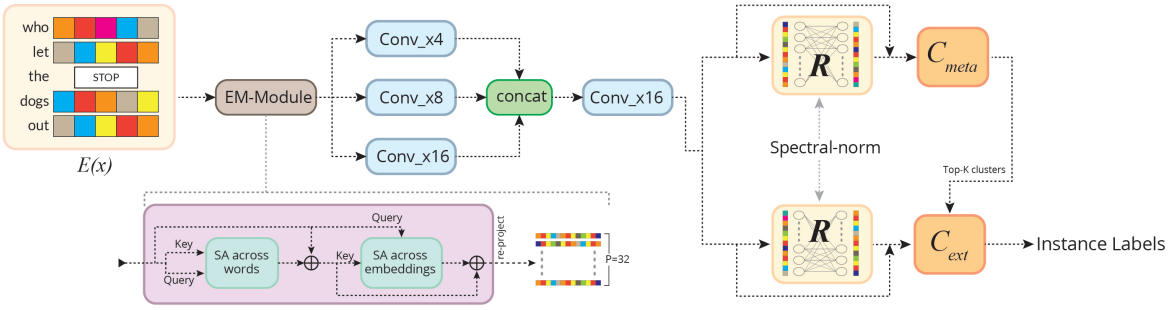

Figure 1: INCEPTIONXML: Model Architecture. The convolution filters on the input data span only a subset of adjacent dimensions in the word embeddings while covering all the input tokens ('*who let the dogs out*'). The Embedding-enhancement module ('EM-module') is shown in detail with its orthogonal self-attention layers followed by a projection layer.

In the context of the aforementioned problems, we hypothesize and empirically demonstrate the suitability of using CNNs to convolve over the embedding dimensions of the inputs instead of the word dimensions for short-text queries.

Embedding Convolutions, by convolving over embeddings in a stacked setting, enable the model to detect correlations or "coupled semantics" between different dimensions in the embedding space by processing a limited subset of semantics at a time. As compared to traditional convolutional operation, embedding convolutions create significantly larger and enriched activation maps for the same inputs, while requiring significantly lesser parameters by using smaller filters of size $\mathbb{R}^{S\times 16}$, where S is the maximum sequence length of the input. We show empirically that this modified approach works well for both short as well as *medium* queries of up to 32 words, significantly outperforming conventional CNN-based approaches (Liu et al., 2017; Kim, 2014) for short-text XML task.

As some readers might rightfully argue, pre-trained word embeddings are typically not trained with any incentive for localizing semantic information in the embedding dimension. This means that the coupled semantics that our model tries to detect may initially occur across spatial distances that are too large for the convolution kernel to detect. To solve this problem, we process the stacked word embeddings with self-attention based embedding enhancement module before applying embedding convolutions. This lets information flow across every pair of semantics irrespective of the spatial distance between them, increasing our convolutional feature extractor's effective receptive field and enabling it to detect the coupled semantics.

4 Proposed Model - InceptionXML

Problem Setup : Given a training set $\{x_i, y_i\}_{i=1}^{N}$, the input instance x_i represents a short-text query, and the corresponding label set is $y_i \in \{0,1\}^L$ where L denotes the total number of labels. It may be noted that even though $L \sim 10^6$, an instance is annotated only with a few (Table 4 in Appendix) positive labels. The goal is to learn a classifier which, for a novel test instance x, predicts the top-k labels towards better precision@k and propensity-scored precision@k (Bhatia et al., 2016).

Towards this goal, the main body of our encoder, shown in Figure 1, consists of three modules that are applied sequentially on the word embeddings. These are (i) an embedding enhancement module, (ii) embedding convolution layers and (iii) an extreme linear classifier.

4.1 Embeddings

As inputs to our model, we stack the word embeddings sequentially. The word embeddings are initialized with d-dimensional pre-trained GloVe embeddings (Pennington et al., 2014) where $d = 300$. Embeddings of words that do not exist in GloVe are initialized with a random vector sampled from the uniform distribution $\mathcal{U}(-0.25, 0.25)$. With NLTK for tokenization, we use white space separated pre-processing function and remove the stop words and punctuation from the raw data (Liu et al., 2017).

4.2 Embedding Enhancement Module

This module enhances the initially stacked word embeddings lacking structure and contextual information, and make it word order agnostic. The module consists of two orthogonal self-attention layers applied sequentially on the word and the em-

bedding dimensions followed by a projection layer, effectively encoding global information both, on a word-level and on a semantic-level (Figure 4).

We use the *SimpleSelfAttention* (Doria, 2019) variant of self-attention in our model, which is a simplification of the corresponding operation described in Zhang et al. (2019). For the key and query matrices X and Y respectively, this is given by the function $G(\cdot,\cdot)$ as follows :

$$\mathrm{G}(X,Y) = \gamma \cdot (XY^T) \cdot f(X) + X$$

The equivalent of the value matrix from Zhang et al. (2019) is represented by $f(X)$ where $f(\cdot)$ is a convolution layer and γ is a learnable scalar parameter. The sequential attention formulation in our embedding enhancement module is given by:

$$\mathrm{SA}_{\text{words}} = \mathrm{G}(E(x),\ E(x)) \ \in \mathbb{R}^{S\times d}$$
$$\mathrm{SA}_{\text{embs}} = \mathrm{G}(\mathrm{SA}_{\text{words}}^T,\ E(x)^T)^T \ \in \mathbb{R}^{S\times d}$$

where $E(x)$ denotes the stacked word embeddings for a sample text input x such that $E(x) \in \mathbb{R}^{S\times d}$. Finally, each dimension of the intermediate embeddings $\mathrm{SA}_{\text{embs}}$ is then projected to a p-dimensional space where $p = 32$ to obtain the final enhanced embeddings SA_{out} where $\mathrm{SA}_{\text{out}} \in \mathbb{R}^{p\times d}$. The information flow across the embeddings in this module followed by per-dimension projection makes SA_{out} independent of the word order in short-text queries and makes our model more robust to their lack of structure.

4.3 Embedding Convolution Layers

We employ three parallel branches of one-dimensional convolution layers V_i, $i \in [1,2,3]$ with filter sizes of w_i where $w_i \in [4,8,16]$ each with a stride of 4 along the embedding dimension and p output channels. Let h_{w_i} be the result of applying V_i over SA_{out}. We concatenate all resultant h_{w_i} row-wise before passing them to the next layer.

$$h_{w_i} = V_i * \mathrm{SA}_{\text{out}}$$
$$h_f = V_f * [h_{w_1}, h_{w_2}, h_{w_3}]$$

A final embedding convolutional layer V_f with kernel size of 16 and stride 4 is applied on the concatenated feature map, which is further flattened to form the final feature representation h_f. This formulation allows V_f to have an effective receptive field spanning $1/4^{th}$ of the enhanced embeddings, further obviating the locality constraints of CNNs as highlighted in section 3.

4.4 Extreme Linear Classifier

The first layer R transforms the feature map from the encoder with a skip-connection while keeping the dimensions same. The next linear layer W has one-vs-all classifiers for each label in the dataset which projects the features to the label space.

$$g = relu(R \cdot h_f) + h_f$$
$$\hat{y} = \sigma(W \cdot g)$$

Loss: We use the binary cross entropy loss to train our model. Here $y \in \{0,1\}^L$ represents the ground-truth multi-hot encoded targets and $\hat{y}$ are the model predictions.

$$\mathrm{BCE}(y,\hat{y}) = -\sum_{j\in L}(1-y_j)\log(1-\hat{y}_j) + y_j \log(\hat{y}_j)$$

5 InceptionXML+ Framework

INCEPTIONXML described previously scales to datasets with hundreds of thousands of labels. However, scaling up to millions of labels in its existing form is difficult as the loss computation in equation above involves calculation of loss over all L labels, a very large majority of which are negative labels for a given instance. We, thus, propose a scalable extension to our encoder, called INCEPTIONXML+, in which the loss is computed over only for the hardest negative labels.

Hard Negative-Mining of Labels: While techniques have been studied for efficient hard-negative label mining under *fixed representation* of data points (Jain et al., 2019; Dahiya et al., 2021b), only recent algorithms (Jiang et al., 2021) have come up with *dynamic* hard negative-mining techniques. For this work, we improve upon the more recent dynamic hard negative-mining strategy to develop an end-to-end trainable framework INCEPTIONXML+. Following the approach popularized by these recent methods, our model makes predictions in two stages: (i) shortlisting K label-clusters or "meta-labels" using a meta-classifier, and (ii) employing a computationally feasible number of one-vs-all classifiers corresponding to the labels included in the shortlisted clusters to get the final predicted labels and perform backpropagation.

Label Clustering To perform label clustering, we construct Probabilistic Label Tree (PLT) using the labels' Positive Instance Feature Aggregation (PIFA) representation over sparse BOW features of their training samples as done in (Jiang et al., 2021; Chang et al., 2020). More specifically, we

use balanced 2-means clustering to recursively partition the label set until we have a mapping C from L labels to L' clusters where $L' \ll L$ (Table:4 in Appendix shows values for L').

Drawbacks of LIGHTXML framework: When scaling our model using dynamic hard-negative mining as done in LIGHTXML (Jiang et al., 2021), we noticed that the performance of our encoder is bottlenecked by a poorly performing meta-classifier. From the training metrics (in Fig: 2), we see a smooth increment in the P@1 values for the extreme classifier (dashed blue) while the meta-classifier is unable to catch-up (dashed red). This indicates that these two sub-tasks are not aligned well enough for the encoder to learn suitable common representations that work well simultaneously for both the sub-tasks. Our observations also indicate the fact that the extreme task is easier to learn on shortlisted labels than the meta-task on label clusters, and the model tends to learn representations that benefit the extreme task at the expense of the meta-task.

We make key changes to the dynamic hard-negative mining framework in accordance with these observations. These changes can be broadly grouped into two sets. The first set of changes consist of architectural changes meant to bring the two tasks closer in order to enable the encoder to learn better common representations. In the second set of changes, we make modifications to the training loop in order to force the encoder to learn representations that improve the performance of the meta-classifier while not compromising on the performance of the extreme task.

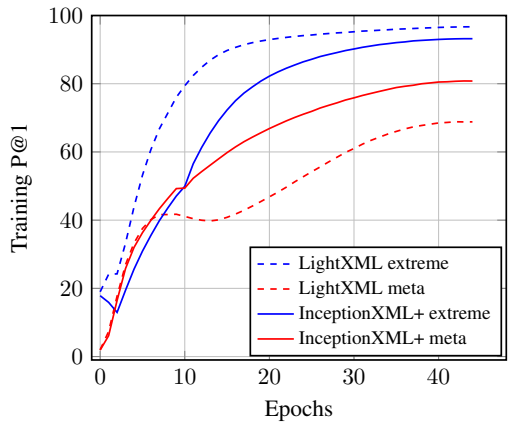

Figure 2: Progress of training (Precision@1) for the extreme and meta-classifier of LIGHTXML and INCEPTIONXML+ framework for AmazonTitles-670K

5.1 Decoupled Architecture

To bring the two prediction tasks closer, we give them similar structures by adding a linear layer with a residual connection before the meta-classifier.

$$g_m = relu(R_m \cdot h_f) + h_f$$
$$\hat{y}_m = \sigma(W_m \cdot g_m)$$
$$\hat{\mathcal{S}} = C^{-1}(\ top_K(\hat{y}_m, k)\ )$$

We create a shortlist $\hat{\mathcal{S}}$ of all the labels in the top K label clusters as predicted by the meta-classifier using a label cluster to label mapping C^{-1}. The extreme classifier then predicts the probability of the query belonging to only these shortlisted labels, instead of all L labels.

$$g_e = relu(R_e \cdot h_f) + h_f$$
$$\hat{y}_{e,l} = \sigma(W_{e,l} \cdot g_e),\ \forall l \in \hat{\mathcal{S}}$$

Architectural similarity of branches alone doesn't ensure strong common representation learning. To help the encoder learn suitable common representations, we further bring the two branches closer by (i) increasing the fan out of label clustering, and (ii) adding spectral norm to the penultimate linear layers in both heads. Increasing the fan out of label clustering brings the meta-task closer to the extreme-task by increasing the "extremeness" of the meta-task. Addition of spectral normalization prevents the weights of the hidden layers of both task heads from drifting too far from each other (Dahiya et al., 2021b). Not only does this heavily improve upon the original implementation of dynamic negative-hard mining framework as proposed in (Jiang et al., 2021), but also inherently combines the task of the two stages of the DeepXML pipeline into an end-to-end trainable model. Even though we observe substantial gains from increasing the fan out, this comes at the cost of making the meta-classifier heavier. So, in practice we aim to strike a balance (Table:2) between number of clusters and model efficiency for non-trivial gains in accuracy.

5.2 Detached Training

To force the encoder to learn representations benefiting the meta-task, we detach i.e. stop the flow of gradients from the extreme classifier head to the encoder (Algorithm 1), for the initial $1/4$-th of the training loop. This results in shortlisting of harder negative labels for the extreme classifier

to learn during training time and ensuring higher recall during inference time.

Detaching instead of simply removing the extreme classification head has the advantage of training the layers in this head. This keeps it in sync with the changing encoder representations without allowing it to affect the training of the meta-classifier. This setting is possible because of the spectral norm (refer Figure 1) applied to the weights of the penultimate layers in both the heads which ensures that the encoder learnt for the meta-task remains relevant for the extreme task when its gradients are re-attached.

Loss: As with INCEPTIONXML, we use BCE loss for training. The losses for the meta-classifier and the extreme classifier are given by:

$$\mathcal{L}_{meta} = \text{BCE}(y_m \ , \ \hat{y}_m),$$
$$\mathcal{L}_{ext} = \text{BCE}(y_{e,l} \ , \ \hat{y}_{e,l}) \ \ \forall l \in \hat{\mathcal{S}}.$$

The final loss for back-propagation is the sum of the above i.e. $\mathcal{L} = \mathcal{L}_{meta} + \mathcal{L}_{ext}$. For prediction, the final ranking is produced by using the logits of the extreme classifier. The complete training algorithm for INCEPTIONXML+ is given below

Algorithm 1: Training algorithm for INCEPTIONXML+

```
for epoch in (1, epochs):
  for x, y in data:
    z = E(x)
    h = encoder(z)
    y_meta = meta_classifier(h)
    y_cluster = label_to_cluster(y)
    meta_loss = bce(y_meta, y_cluster)

    # shortlisting top K clusters
    top_k = get_top_K_clusters(y_meta, k)
    candidates = cluster_to_label(top_k)
    # add missing positive labels
    candidates = add_missing(candidates,y)

    # detached training
    if epoch <= epochs/4:
      h = h.detach()
    y_ext = ext_classifier(h, candidates)
    ext_loss = bce(y_ext, y, candidates)
    loss = meta_loss + ext_loss
    loss.backward()

    # gradient descent
    update(E, encoder, meta_classifier,
     ext_classifier)
```

6 Experiments

Datasets: We evaluate the proposed INCEPTIONXML(+) frameworks on 4 publicly available benchmarks from the extreme classification repository (Bhatia et al., 2016). The details of the datasets are given in Table 4 (Appendix), the number of labels range from 350,000 (WikiSeeAlsoTitles-350K) to 2.8 Million (AmazonTitles-3M). Evaluation on the Wikipedia datasets involves predicting tags and related pages from Wikipedia page titles and Amazon datasets involves predicting items frequently bought together from just product names.

Method	P@1	P@3	P@5	PSP@1	PSP@3	PSP@5
	AmazonTitles-670K					
INCEPTIONXML+	41.48	**37.49**	**34.60**	26.78	30.15	**33.31**
INCEPTIONXML	**41.83**	37.46	34.10	**28.17**	**30.98**	33.28
ASTEC	39.97	35.73	32.59	27.59	29.79	31.71
ASTEC-3	40.63	36.22	33.00	28.07	30.17	32.07
LIGHTXML	41.55	37.31	34.10	25.23	28.79	31.92
APLC-XLNET	34.55	30.58	27.49	19.82	22.22	24.19
ATTENTIONXML	37.92	33.73	30.57	24.24	26.43	28.39
XML-CNN	35.02	31.37	28.45	21.99	24.93	26.84
DISMEC	38.12	34.03	31.15	22.26	25.45	28.67
PARABEL	38.00	33.54	30.10	23.10	25.57	27.61
BONSAI	38.46	33.91	30.53	23.62	26.19	28.41
MACH	34.92	31.18	28.56	20.56	23.14	25.79
	WikiSeeAlsoTitles-350K					
INCEPTIONXML+	20.51	14.85	11.79	10.11	12.45	14.44
INCEPTIONXML	**21.50**	**15.18**	**11.98**	**10.90**	**13.04**	**14.93**
ASTEC	20.42	14.44	11.39	9.83	12.05	13.94
ASTEC-3	20.61	14.58	11.49	9.91	12.16	14.04
LIGHTXML	20.18	13.59	10.54	8.52	10.04	11.53
APLC-XLNET	19.80	13.80	10.86	7.29	9.31	11.01
ATTENTIONXML	15.86	10.43	8.01	6.39	7.20	8.15
XML-CNN	17.75	12.34	9.73	8.24	9.72	11.15
DISMEC	16.61	11.57	9.14	7.48	9.19	10.74
PARABEL	17.24	11.61	8.92	7.56	8.83	9.96
BONSAI	17.95	12.27	9.56	8.16	9.68	11.07
MACH	14.79	9.57	7.13	6.45	7.02	7.54
	WikiTitles-500K					
INCEPTIONXML+	44.93	**30.85**	**22.01**	18.81	20.55	21.07
INCEPTIONXML	**47.28**	27.14	19.39	**20.79**	**21.01**	**21.17**
ASTEC	46.01	25.62	18.18	18.62	18.59	18.95
ASTEC-3	46.60	26.03	18.50	18.89	18.90	19.30
LIGHTXML	36.16	17.54	12.03	9.33	7.81	7.60
APLC-XLNET	41.84	21.87	15.53	14.59	12.95	13.15
ATTENTIONXML	42.89	22.71	15.89	15.12	14.32	14.22
XML-CNN	43.45	23.24	16.53	15.64	14.74	14.98
DISMEC	39.89	21.23	14.96	15.89	15.15	15.43
PARABEL	42.50	23.04	16.21	16.55	16.12	16.16
BONSAI	42.60	23.08	16.25	17.38	16.85	16.90
MACH	33.74	15.62	10.41	11.43	8.98	8.35
	AmazonTitles-3M					
INCEPTIONXML+	46.65	45.16	**43.45**	**16.18**	**19.19**	**21.28**
INCEPTIONXML	-	-	-	-	-	-
ASTEC	47.64	44.66	42.36	15.88	18.59	20.60
ASTEC-3	**48.74**	**45.70**	43.31	16.10	18.89	20.94
LIGHTXML	-	-	-	-	-	-
APLC-XLNET	-	-	-	-	-	-
XML-CNN	-	-	-	-	-	-
ATTENTIONXML	46.00	42.81	40.59	12.81	15.03	16.71
DISMEC	41.13	38.89	37.07	11.98	14.55	16.42
PARABEL	46.42	43.81	41.71	12.94	15.58	17.55
BONSAI	46.89	44.38	42.30	13.78	16.66	18.75
MACH	37.10	33.57	31.33	7.51	8.61	9.46

Table 1: Comparison of InceptionXML to state-of-the-art extreme classification algorithms on benchmark datasets. The best-performing approach is in **bold** and the second best is underlined. '-' infront of a model implies that the model doesn't scale for that dataset.

6.1 Main Results

The main results of our experiments are shown in Table 1. For most of the dataset-metric combinations, the proposed models, INCEPTIONXML and INCEPTIONXML+, not only outperform the previous state-of-the-art ASTEC and but also its ensemble version ASTEC-3 with non-trivial gains. Notably, INCEPTIONXML gains an average of 4.2% and 8.18% over ASTEC on all three datasets except AmazonTitles-3M as measured by the P@1 and PSP@1 metrics. Furthermore, the following observations can be made :

- The proposed models achieves at least 10% relative improvement as compared to XML-CNN (Liu et al., 2017), which captures n-grams for representation learning and the RNN-based ATTENTIONXML (You et al., 2019).
- Significant gains (upto 20% in some cases) are obtained compared to transformer-based models such as LIGHTXML (Jiang et al., 2021), and APLC-XLNET (Ye et al., 2020). Notably, none of these architectures scale to AmazonTitles-3M dataset, demonstrating the efficacy and scalability of the proposed light-weight encoder.
- Our models also significantly outperform non-deep learning approaches using bag-of-words representations such as the label-tree based algorithms like BONSAI (Khandagale et al., 2020) and PARABEL (Prabhu et al., 2018), and DISMEC (Babbar and Schölkopf, 2017).
- We note that INCEPTIONXML generally outperforms INCEPTIONXML+ on several benchmarks, especially for the PSP metrics. We attribute this to the fact that INCEPTIONXML always gets information about all negative labels instead of only hard-negative labels. This allows it to perform better on tail labels for which the label clusters in INCEPTIONXML+ may not be optimal.

6.2 Ablation Results

Permuting Embedding Dimension: To show that INCEPTIONXML is independent of the order of embedding dimensions, we randomly permute the dimensions of the input word embeddings before start of the training, train with this fixed permuted order and evaluate in the standard manner. This is repeated 10 times with different permutations before training. Only *slight* variation in performance metrics can be observed in figure 3 with respect to the median of each boxplot which implies that the order of embedding dimensions has *little or no* impact over the results of our model.

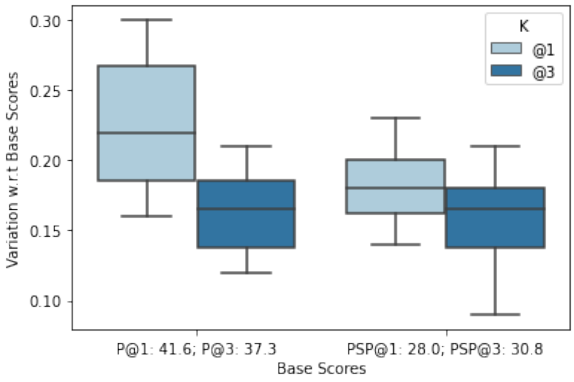

Figure 3: Variation in scores after shuffling embedding dimensions randomly before start of training for AmazonTitles-670K dataset. The boxplot only shows a variation in the performance metrics from the 10 runs. Different scores and statistics can be obtained by adding the values in the y-axis to the base scores on the x-axis.

InceptionXML+: Table 2 shows a comparison of the proposed INCEPTIONXML+ pipeline vis-à-vis LIGHTXML for AmazonTitles-670K dataset. It is clear that the INCEPTIONXML+ framework significantly improves upon the dynamic hard-negative mining technique as proposed in LIGHTXML in terms of performance in both P@K and PSP@K metrics. While we notice consistent improvement for our decoupled architecture (even without detached training) as the fanout is increased for the label clustering step, the results of our encoder in the LIGHTXML framework results only show marginal improvement and a dip later across all

(L', T_K)	Model	P@1	P@5	PSP@1	PSP5
8K, 200	Ours	40.37	33.01	26.02	31.61
	Ours w/o Detaching	40.41	33.06	25.95	31.63
	in LightXML Framework	39.60	32.54	25.14	30.78
16K, 400	Ours	40.99	33.63	26.36	32.31
	Ours w/o Detaching	41.01	33.61	26.30	32.23
	in LightXML Framework	39.87	32.86	25.07	31.00
32K, 800	Ours	41.42	34.23	26.64	32.87
	Ours w/o Detaching	41.26	34.13	26.52	32.77
	in LightXML Framework	39.98	33.11	24.73	31.01
65K, 1600	Ours	**41.48**	**34.60**	26.78	**33.31**
	Ours w/o Detaching	41.03	34.23	26.49	32.89
	in LightXML Framework	39.67	33.06	23.92	30.64
–, –	in DeepXML Pipeline	38.53	32.21	**27.80**	31.62

Table 2: Ablation results on AmazonTitles-670K for the impact of increasing fan-out of label clustering (L') for our encoder in different scaling up frameworks where T_K represents the number of Top K shortlisted clusters. (*Ours* - InceptionXML+ Framework)

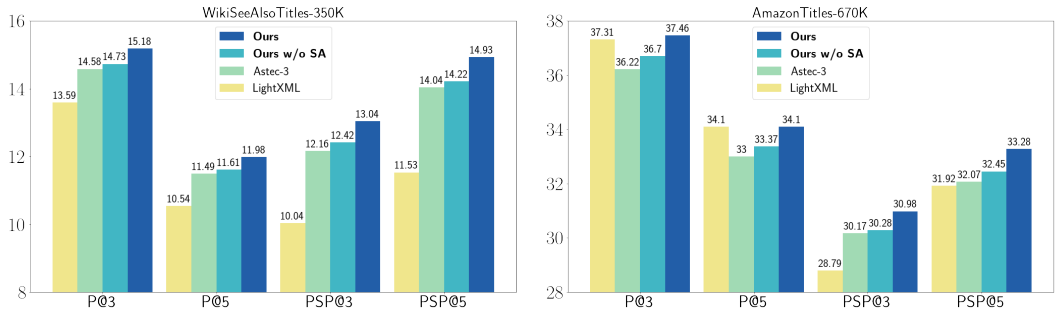

Figure 4: Performance with and without self-attention layers on WikiSeeAlsoTitles-350K & AmazonTitles-670K

metrics. Note that we keep the shortlisted labels consistent by doubling the number of shortlisted meta-labels as the fan-out doubles. It may be also be noted that as the fan-out increases, our detached training method improves the results more prominently. This can be attributed to the fact that we bring the two tasks closer by increasing the fan-out and the representations learnt by the encoder for the meta-task become increasingly more relevant to the extreme-task when the gradients of the extreme classifier are re-attached during training.

Test data	P@3	P@5
Original AmazonTitles-670K	37.49	34.60
Permuted Word-order	36.78 ± 0.05	33.92 ± 0.03
Original WikiSeeAlsoTitles-350K	14.85	11.79
Permuted Word-order	14.50 ± 0.03	11.51 ± 0.02

Table 3: Comparison of results with original test data and that obtained by permuting the word order in the test set for INCEPTIONXML+

Robustness to Lack of Word-order: For testing the robustness of our method to the order of words in input data, we train the InceptionXML+ on the original training data for AmazonTitles-670K, but randomly permute the words in test set, and evaluate the performance. This is repeated 10 times with different test set permutations (Table 3). We witness only a minor dip in performance across the metrics still outperforming ASTEC-3 and demonstrating the robustness of our encoder to lack of structure in short-text queries.

Embedding Enhancement Module: The sequentially applied self-attention layers improve INCEPTIONXML's performance by 2-4% on the performance metrics as shown in figure 4. However, the superior representation learning capability of our encoder for short-text queries are further demonstrated in Figure 4 as even without the self-attention layers, our model outperforms the ensemble model ASTEC-3 and LIGHTXML.

InceptionXML in DeepXML Framework: We integrate our encoder with the DeepXML (Dahiya et al., 2021b) pipeline as used by ASTEC and find it inflexible to improve upon due to the requirement of fixed representations for their label shortlisting strategy. Moreover, when using our encoder as a drop-in replacement, we find our encoder's performance degrades in terms of precision in the DeepXML Framework as compared to the performance in the vanilla LIGHTXML Framework (Table2: last row). This indicates the overall advantage of using dynamic hard-negative mining compared to techniques requiring fixed representations.

6.3 Training Time and Model Size

Our model's training time ranges from 7 hours with INCEPTIONXML on the WikiSeeAlsoTitles-350K dataset to 31 hours on AmazonTitles-3M. We observe ~40% improvement in training time by using the INCEPTIONXML+ pipeline compared to the INCEPTIONXML. Furthermore, INCEPTIONXML is extremely lightweight in terms of model size containing only 400K parameters while INCEPTIONXML+ contains only 630K parameters.

7 Conclusion

In this work, we revisited the architecture of convolution networks for the task of short-text extreme classification. We recast the conventional convolutional architecture to capture coupled semantics along the embedding dimensions. Augmented with a self-attention based order-agnostic Embedding Enhancement module, we show that the proposed approach leads to a light-weight encoder which betters state-of-the-art performance on 4 benchmark datasets. By addressing the shortcomings of the training regimen of LIGHTXML, we also develop an extension to our model - INCEPTIONXML+ that scales to dataset with 3 millions labels.

References

R. Babbar and B. Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-label Classification. In *WSDM*.

K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code.

K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. 2015. Sparse Local Embeddings for Extreme Multi-label Classification. In *NIPS*.

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, and Ion Androutsopoulos. 2019. Large-scale multi-label text classification on eu legislation. *arXiv preprint arXiv:1906.02192*.

W-C. Chang, H.-F. Yu, K. Zhong, Y. Yang, and I. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *KDD*.

Wei-Cheng Chang, Daniel Jiang, Hsiang-Fu Yu, Choon-Hui Teo, Jiong Zhang, Kai Zhong, Kedarnath Kolluri, Qie Hu, Nikhil Shandilya, Vyacheslav Ievgrafov, et al. 2021. Extreme multi-label learning for semantic matching in product search. *arXiv preprint arXiv:2106.12657*.

K. Dahiya, A. Agarwal, D. Saini, K. Gururaj, J. Jiao, A. Singh, S. Agarwal, P. Kar, and M. Varma. 2021a. Siamesexml: Siamese networks meet extreme classifiers with 100m labels. In *Proceedings of the International Conference on Machine Learning*.

K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. 2021b. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *WSDM*.

Sebastian Doria. 2019. Simple self-attention https://github.com/sdoria/simpleselfattention.

H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. 2019. Slice: Scalable Linear Extreme Classifiers trained on 100 Million Labels for Related Searches. In *WSDM*.

H. Jain, Y. Prabhu, and M. Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking and Other Missing Label Applications. In *KDD*.

Ting Jiang, Deqing Wang, Leilei Sun, Huayi Yang, Zhengyang Zhao, and Fuzhen Zhuang. 2021. Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7987–7994.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

S. Khandagale, H. Xiao, and R. Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning*, 109(11):2099–2119.

Y. Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*.

J. Liu, W. Chang, Y. Wu, and Y. Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *SIGIR*.

A. Mittal, K. Dahiya, S. Agrawal, D. Saini, S. Agarwal, P. Kar, and M. Varma. 2021a. Decaf: Deep extreme classification with label features. In *Proceedings of the ACM International Conference on Web Search and Data Mining*.

A. Mittal, N. Sachdeva, S. Agrawal, S. Agarwal, P. Kar, and M. Varma. 2021b. Eclare: Extreme classification with label graph correlations. In *Proceedings of The ACM International World Wide Web Conference*.

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

Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *WWW*.

Kshitij Tayal, Nikhil Rao, Saurabh Agarwal, Xiaowei Jia, Karthik Subbian, and Vipin Kumar. 2020. Regularized graph convolutional networks for short text classification. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 236–242.

Jin Wang, Zhongyuan Wang, Dawei Zhang, and Jun Yan. 2017. Combining knowledge with deep convolutional neural networks for short text classification. In *IJCAI*.

Zhongyuan Wang and Haixun Wang. 2016. Understanding short texts. In *the Association for Computational Linguistics (ACL) (Tutorial)*.

H. Ye, Z. Chen, D.-H. Wang, and Davison B. D. 2020. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification. In *ICML*.

R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. 2019. Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *Neurips*.

Hsiang-Fu Yu, Kai Zhong, and Inderjit S Dhillon. 2020. Pecos: Prediction for enormous and correlated output spaces. *arXiv preprint arXiv:2010.05878*.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. 2019. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR.

A Appendix

A.1 Dataset Details

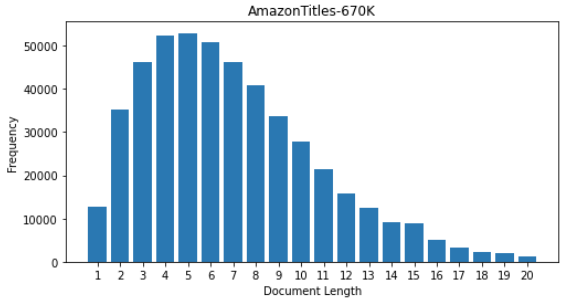

Figure 5: Sequence lengths of the input instance plotted against corresponding frequency for AmazonTitles-670K dataset. For this dataset, 70% of training instances have ≤ 8 words, and 30% have ≤ 4 words.

Figure 5 above details the distribution of sequence lengths in AmazonTitles-670K dataset. Also, among the other key features, such as # of training/test instances and labels, Table 4 (in Appendix below) confirms the short-text nature of these datasets. The last three columns give the hyper-parameter values for the clustering step used in the INCEPTIONXML+.

A.2 Vocabulary & Word Embedding

As opposed to taking their TF-IDF weighted linear combination as used in some recent works (Dahiya et al., 2021b,a; Mittal et al., 2021a) or the more conventional bag-of-words representations approaches like (Babbar and Schölkopf, 2017; Prabhu et al., 2018), we use the approach of stacking Glove embeddings (Pennington et al., 2014) as done in (Kim, 2014; Liu et al., 2017; Wang et al., 2017).

For a fair comparison, we use exact same size of vocabulary space as (Dahiya et al., 2021b) for all benchmark datasets. As state before, we use wide-space tokenizer and find empirically that our model works better without using sub-word tokenizers like word-piece or sub-word based embeddings like fastText (Joulin et al., 2016).

A.3 Evaluation Metrics

As stated earlier, the main application of short-text XML framework is in recommendation systems and web-advertising, where the objective of an algorithm is to correctly recommend/advertise among the top-k slots. Thus, for evaluation of the methods, we use precision at k (denoted by $P@k$), and its propensity scored variant (denoted by $PSP@k$) (Jain et al., 2016). These are standard and widely used metrics by the XML community (Bhatia et al., 2016).

For each test sample with observed ground truth label vector $y \in \{0,1\}^L$ and predicted vector $\hat{y} \in \mathbb{R}^L$, $P@k$ is given by :

$$P@k(y,\hat{y}) := \frac{1}{k} \sum_{\ell \in top@k(\hat{y})} y_\ell$$

where $top@k(\hat{y})$ returns the k largest indices of $\hat{y}$.

Since $P@k$ treats all the labels equally, it doesn't reveal the performance of the model on tail labels. However, because of the long-tailed distribution in extreme classification datasets, one of the main challenges is to predict tail labels correctly, which are more valuable and informative compared to head classes, and it is essential to measure the performance of the model specifically on tail labels. By alluding to the phenomenon of missing labels in the extreme classification setting and its relation to tail-labels, $PSP@k$ was introduced in Jain et al. (2016) as an unbiased variant of original precision at k under no missing labels. This is widely used by the community to compare the relative performance of algorithms on tail-labels, and is also another metric used in our relative comparisons among various extreme classification algorithms in Tables 1 and 2 for main results and ablation tests respectively.

B Responsible NLP Research Checklist

B.1 Limitations

- Given that the convolution operation spans over the entire document length, the proposed method is suited for short and medium length text sequences.
- Our method is agnostic to the presence of label texts, which despite constraining the problem to a much smaller subset, have been shown to help in achieving better prediction performance.

B.2 Potential Risks

We do not forsee any potential risks of our methods. Rather, it should be seen to be as energy-efficient alternatives to large-transformer models for the core textual and language problems encountered in search and recommendation.

Datasets	# Features	# Labels	# Training	# Test	APpL	ALpP	#W≤ 4	#W≤ 8	L'	Top K	ALpC
WikiSeeAlsoTitles-350K	91,414	352,072	629,418	162,491	5.24	2.33	82%	98%	32K	800	10
WikiTitles-500K	185,479	501,070	1,699,722	722,678	23.62	4.89	83%	98%	65K	1200	8
AmazonTitles-670K	66,666	670,091	485,176	150,875	5.11	5.39	40%	70%	65K	1600	11
AmazonTitles-3M	165,431	2,812,281	1,712,536	739,665	31.55	36.18	15%	52%	131K	1000	22

Table 4: Dataset Statistics. APpL denotes the average data points per label, ALpP the average number of labels per point. #W is the number of words in the training samples. For our scaled up model, L' and Top K denote the number of label-clusters and the value of K for top K clusters chosen per dataset while ALpC denotes the average labels per cluster.