Extreme Multi-Label Text Classification

Prakash Nath Jha (2018201013)
Chittaranjan Rath (2018201007)
Nitish Srivastava (2018201012)

Abstract

In Extreme multi-label text classification (XMTC) the task is to tag each given text with its most relevant labels (multiple labels) from an extremely large-scale label set. Number of labels can range from several thousands to few hundred millions, which makes it difficult for traditional machine learning based methods like SVM to classify the labels. Also handling tail labels is a big challenge when we have millions of labels. In this project we first study *AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification* paper which proposes a multi label attention mechanism which captures the most relevant part of the text corresponding to each label. The paper also proposes a shallow and wide PLT which helps to tackle the problems of extreme multi-label classification with millions of labels. We analyse the code and train it on EUR-Lex, Wiki10-31K and Amazon-670K dataset. We also extract attention score and attention weight from these models. We then implement *HAXMLNet: Hierarchical Attention Network for Extreme Multi-Label Text Classification* paper which is a more scalable version of AttentionXML paper. We train our model for EUR-Lex, Wiki10-31K dataset and compare their performance with AttentionXML paper.

1 Introduction

In Extreme multi-label text classification (XMTC) the task is to tag each given text with its most relevant labels (multiple labels) from an extremely large-scale label set. For example Amazon-670K contains 667317 as number of labels ,EUR-Lex has 3801 labels. Various methods of XMTC can be classified into :

- 1-v/s-All
- Embedding based
- Label Tree based
- Deep Learning based

In this paper the relevant methods are deep learning-based and label tree-based methods.

Deep Learning based

- **XML-CNN :** It uses a convolutional neural network (CNN) and dynamic pooling to learn the text representation. Since the same text representation is given for all labels it can't capture which parts of the text is important for each label.
- **Seq2Seq :** It uses a RNN to encode a given raw text and attentive decoder to predict labels. Since there is no relation between labels which are predicted so this sequential prediction is counter intuitive.

Label Tree based

- **Parabel :** It is a state of the art label tree based method which uses bag-of-words (BOW) features.It creates a balanced binary label tree by dividing the nodes into two balanced clusters until the cluster size doesn't exceed a threshold value (e.g. 100).
- **ExtremeText :** It is based on FastText and uses dense features instead of BOW. It ignores the order of words without considering context information.

2 Literature survey

2.1 AttentionXML

The steps of AttentionXML include

1. Building a shallow and wide PLT
2. Using multi-label attention and BiLSTM ,for each level d except the first level of the PLT train an attention aware deep model.

2.2 Building Shallow and Wide PLT

Each node n has a value $z_n \in \{0, 1\}$. This value is different for each sample and it indicates if the subtree rooted at this level has a leaf (original label) relevant to this sample. A shallow and wide tree with depth H and width M is built as the *deep* tree (with a high tree depth) for an extreme scale dataset, deteriorates the performance due to an inaccurate approximation of likelihood, and the accumulated and propagated errors along the tree. Initial tree is built using top-down hierarchical clustering with a small cluster size M.The labels are then recursively partitioned into two smaller clusters, which correspond to internal tree nodes, by a balanced k-means (k=2) until the number of labels smaller than M(max_leaf configuration in the yaml file). This tree is then compressed into a shallow and wide PLT by removing nodes between S_{h-1} and S_h and reset nodes between S_h and S_{h-1}. Hence , we are actually saving the different levels as mentioned in the configuration file and there by compressing the tree.

2.3 Learning AttentionXML

Since nodes at a deeper level have less positive samples , training a deep model is ineffective. Hence Attention XML is trained level wise.

1. AttentionXML trains a single deep model for each level of a given PLT in a top-down approach
2. AttentionXML$_d$ for the d-th level ($d > 1$) of the given PLT is only trained by candidates g(x) for each sample x i.e. we sort nodes of the $d-1$-th level level by z_n and then their scores predicted by AttentionXML$_{d-1}$ in the descending order.We keep the top C nodes at the $d-1$-th level and choose their children as g(x).
3. During prediction, for the i-th sample, the predicted score $\hat{y}_{ij}$ for j-th label is computed.Beam Search is used for prediction efficiency.

Attention-aware deep model in AttentionXML consists of five layers:

- *Word Representation Layer* : Pre-trained 300-dimensional GloVe word embedding was used
- *Bidirectional LSTM Layer* : RNNs have a problem called gradient vanishing and exploding during training so LSTM was used . A Bidirectional LSTM (BiLSTM) was used.It involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second.

- *Multi-label Attention Layer* : AttentionXML computes the (linear) combination of context vectors h^i for each label through a multi-label attention mechanism.
- *Fully Connected Layer and Output Layer* : The same parameter values are used for all labels at the fully connected (and output) layers, to emphasize differences of attention among all labels.

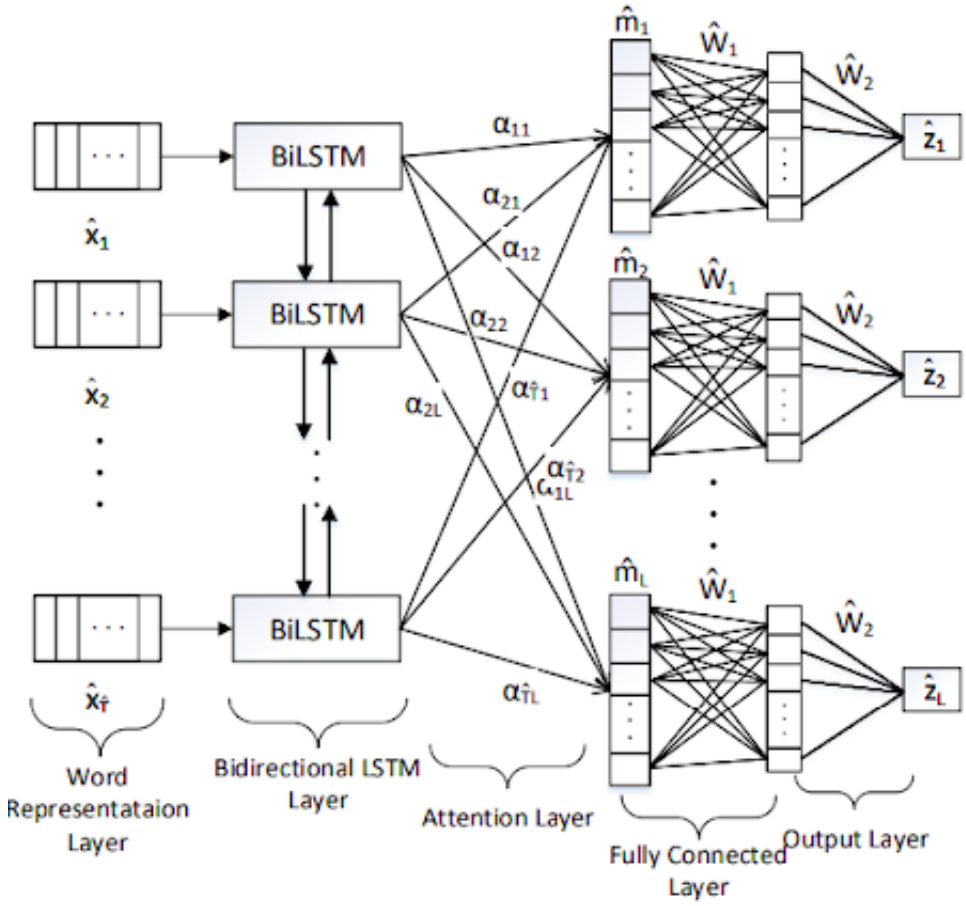

Figure 1: Attention XML Layers

3 Research Methods

HAXML Algorithm:

1. We train a shallow wide PLT for the dataset to generate clusters/groups of labels. Group size and number of groups are hyper parameters which need to be tuned based on a dataset for optimal performance.
2. Generate dataset where labels are replaced by group id based on the PLT trained in level 1.
3. Train AttentionXML model on the modified dataset and store the groups scores. We call this model as HAXML Net G model, as its trained on groups not label.
4. Replace groups with labels for each datapoint. We take all labels including positive labels and negative labels present in each group. This is done for each data point. These labels are termed as candidate labels. These negative labels help in preventing models from overfitting, acting as a regularization.

5. Train another AttentionXML models on this dataset which contains candidate dataset, and store scores for labels corresponding to each data point. We call this model the HAXML Net L model.

6. During prediction step we combine group score and label score coming from HAXML Net G and HAXML Net L respectively for each data point and take top k labels based on combined scores. To get a final score for each label corresponding to every data point we multiply the label score with their corresponding group score. For a given sample, the predicted score $\hat{y}_j$ for jth label based on probability chain rule is as follows:

$$\hat{y}_j = \hat{y}^{group}_{G(j)} \times \hat{y}^{label}_j$$

where $\hat{y}^{group}_{G(j)}$ is the predicted score for the group G(j) by AttentionXMLG, and $\hat{y}^{label}_j$ is the predicted score for the j^{th} label by AttentionXML-L.

7. We take *top k* labels as our final prediction. We then find *P@K* and *nDCG@K* metrics for the test dataset.

4 Dataset

Dataset	**Train Sample**	**Test Sample**	**Labels**
EUR-Lex	15449	3865	3801
Wiki10-31K	14146	6616	29947
Amazon-670K	490449	153025	667317

Table 1: Dataset

Link to download the dataset[3].

5 Evaluation Metrices

We have used P@k (Precision at k) and nDCG@k (normalized Discounted Cumulative Gain at k) as our evaluation metrics for performance comparison, since both P@k and nDCG@k are widely used for evaluation methods for multi-label classification problems.

$$P@k = \frac{1}{k} \sum_{l=1}^{k} \mathbf{y}_{rank(l)}$$

where $\mathbf{y} \in \{0,1\}^L$ is the true binary vector, and rank(l) is the index of the l^{th} highest predicted label. $nDCG@k$ is defined as follows:

$$DCG@k = \sum_{l=1}^{k} \frac{\mathbf{y}_{rank(l)}}{\log(l+1)}$$

$$iDCG@k = \sum_{l=1}^{\min(k, \|\mathbf{y}\|_0)} \frac{1}{\log(l+1)}$$

$$nDCG@k = \frac{DCG@k}{iDCG@k}$$

6 Findings and Analysis

- **Extracted Attention score corresponding to each data point**
  - EUR-Lex : [3801, 500]
  - Wiki10-31K: [29947, 500]
- **Extracted Attention weight for Attention XML Model**
  - EUR-Lex : [3801, 512]
  - Wiki10-31K: [29947, 512]
- **Extracted Attention weight for Fast Attention XML Model**
  - Amazon-670K: [667317, 1024]
- **Network Structure Comparison**

AttentionXML	**Dimensions**
emb.emb.weight	(166402, 300)
lstm.initstate	(4, 1, 256)
lstm.lstm.weight_ih_l0	(1024, 300)
lstm.lstm.weight_hh_l0	(1024, 256)
lstm.lstm.bias_ih_l0	(1024,)
lstm.lstm.bias_hh_l0	(1024,)
lstm.lstm.weight_ih_l0_reverse	(1024, 300)
lstm.lstm.weight_hh_l0_reverse	(1024, 256)
lstm.lstm.bias_ih_l0_reverse	(1024,)
lstm.lstm.bias_hh_l0_reverse	(1024,)
attention.attention.weight	(3801, 512)
linear.linear.0.weight	(256, 512)
linear.linear.0.bias	(256,)
linear.output.weight	(1, 256)
linear.output.bias	(1,)

Table 2: Attention XML Network Structure For EUR-Lex Dataset

From the above network structure (Table 2 and Table 3) we can see that first we have an embedding layer where each token is represented with 300 dimensional Glove embedding.166402 is the vocab size for EUR-Lex dataset. Next we have Bi-Directional LSTM layer, which takes input from embedding layer and learns the bi-directional context of the input corresponding to labels. Between LSTM layer and linear layer we have attention layer of dimension [labels,512]. This attention layer provides attention scores to each labels corresponding to each data points of dimension [labels, 500]. After attention layer we have linear layer and output layer with sigmoid which produces score for each label corresponding to each data point. From the network structure of FastAttentionXML (Table 3) and AttentionXML (Table 2) we can see some structural difference like in FastAttentionXML we have two linear layer and one output layer after LSTM layer, but in AttentionXML we have one linear layer and one output layer.

Based on a flag *parallel_attn* attention weight is distributed across multiple GPUs (here 3) for computational efficiency, and hence its network structure is slightly different than normal FastAttentionXML network. To get the final attention weight we need to concatenate all their attention weight layers; the parallel_attn flag is set true based on a threshold on the number of labels. Also during the PLT building phase it is ensured that the tree is balanced and for that *max_leaf* parameter is important, as if we set *max_leaf* as 1 tree may not be balanced. So *max_leaf* parameter is highly dataset specific, and it should be set such that the tree is balanced.

Fast AttentionXML	Fast AttentionXML with Distributed Attention Weights	Dimensions
-	AttentionWeights.emb.0.weight	(222439,1024)
-	AttentionWeights.emb.1.weight	(222439,1024)
-	AttentionWeights.emb.2.weight	(222439,1024)
Network.emb.emb.weight	Network.emb.emb.weight	(500000, 300)
Network.lstm.init_state	Network.lstm.init_state	(4, 1, 512)
Network.lstm.lstm. weight_ih_l0	Network.lstm.lstm. weight_ih_l0	(2048, 300)
Network.lstm.lstm. weight_hh_l0	Network.lstm.lstm. weight_hh_l0	(2048, 512)
Network.lstm.lstm.bias_ih_l0	Network.lstm.lstm.bias_ih_l0	(2048,)
Network.lstm.lstm.bias_hh_l0	Network.lstm.lstm.bias_hh_l0	(2048,)
Network.lstm.lstm. weight_ih_l0_reverse	Network.lstm.lstm. weight_ih_l0_reverse	(2048, 300)
Network.lstm.lstm. weight_hh_l0_reverse	Network.lstm.lstm. weight_hh_l0_reverse	(2048, 512)
Network.lstm.lstm. bias_ih_l0_reverse	Network.lstm.lstm. bias_ih_l0_reverse	(2048,)
Network.lstm.lstm. bias_hh_l0_reverse	Network.lstm.lstm. bias_hh_l0_reverse	(2048,)
Network.attention.attention. weight	-	(16384, 1024)
Network.linear.linear.0.weight	Network.linear.linear.0.weight	(512, 1024)
Network.linear.linear.0.bias	Network.linear.linear.0.bias	(512,)
Network.linear.linear.1.weight	Network.linear.linear.1.weight	(256, 512)
Network.linear.linear.1.bias	Network.linear.linear.1.bias	(256,)
Network.linear.linear.output.weight	Network.linear.linear.output.weight	(1, 256)
Network.linear.linear.output.bias	Network.linear.linear.output.bias	(1,)

Table 3: Network structure comparsion for Amazon-670K dataset at different levels

Drive Link For Attention Weight and Score : `https://drive.google.com/open?id=1MQPfrf3yiufJxHC3qsH-Ts3cu9fHzdST`

7 Results and Comparison

Dataset	P@1	P@3	P@5	DCG@3	DCG@5
EUR-Lex	87.09	74.02	61.10	77.78	67.45
Wiki10-31K	84.78	70.13	60.25	73.81	65.02

Table 4: HAXML Net Evaluation Scores

Methods	P@1	P@3	P@5	nDCG@3	nDCG@5
AttentionXML	87.47	78.48	69.37	80.61	73.79
HAXML NET	84.78	70.13	60.25	73.81	65.02

Table 5: Wiki10-31K

From table 4 we can observe the scores for HAXML NET model on EUR-Lex and Wiki10-31K dataset. We can immediately observe performance dip for the larger dataset, i.e. Wiki10-31K dataset. For EUR-Lex dataset we observe from table 6 that P@K, nDCG@K scores are almost similar to what is present in AttentionXML paper.P@3 score for HAXML NET is slightly better

Methods	P@1	P@3	P@5	nDCG@3	nDCG@5
AttentionXML	87.12	73.99	61.93	77.44	71.53
HAXML NET	87.09	74.02	61.10	77.78	67.45

Table 6: EURLex

than AttentionXML. For Wiki10-31K dataset from table 5 we observe performance of HAXML NET is lower than that of AttentionXML paper which suggests that performance of HAXML NET decreases with larger dataset. Hyper-parameters for HAXML NET paper like number of groups, groups size tuning plays an important role in increasing score.

AttentionXML	Height	P@1	P@3	P@5	nDCG@3	nDCG@5
No PLT	0	81.26	66.49	39.89	70.29	53.44
Shallow PLT	2	78.57	64.18	38.50	67.84	51.57

Table 7: EUR-Lex

AttentionXML	Height	P@1	P@3	P@5	nDCG@3	nDCG@5
No PLT	0	85.56	77.11	65.48	75.62	69.02
Shallow PLT	2	80.62	72.23	41.74	72.23	52.21

Table 8: Wiki10-31K

From the tables (Table 7 and 8 respectively) above we can see that increasing height(H) of PLT leads to slight decrease in performance. We found from our experiments that performance can be improved if we decrease height and keep *max_leaf* hyper-parameter as low as possible ensuring we build a balanced tree. But it increases time and memory requirements. So choosing these hyper-parameters is a trade-off between performance and time/memory cost.

8 Limitations

In the AttentionXML model we have Bi-Directional LSTM layer which makes it difficult to parallelize. Also training AttentionXML model without PLT on larger datasets like Amazon-670K dataset is not feasible as the models does not scale with the size of the dataset. To deal with it we use AttentionXML model with shallow and wide PLT where we train AttentionXML model at each layer instead of training only AttentionXML model, which is named as FastAttentionXML model. This help in dealing with larger dataset like Amazon-670K dataset, but it increases time complexity of the model as we now need to train more that one model. HAXML Net models tries to address scalability issues of AttentionXML model, but the performance of HAXML Net model is not at par with AttentionXML model or Fast AttentionXML model.

9 Conclusion

In this project we implemented HAXMLNet paper and executed AttentionXML paper for various datasets as mentioned above. The results of our implementation of HAXMLNet for EUR-Lex, Wiki10-31K dataset was found to be comparable with the results in the AttentionXML paper for various evaluation metrics and it was also comparable for results of various other methods mentioned in the AttentionXML paper.

Code Link: `https://github.com/misterpawan/MTP2020-AttentionXML`

10 References

[1] Ronghui You ,Zihan Zhang ,Ziye Wang ,Suyang Dai, Hiroshi Mamitsuka ,Shanfeng Zhu, AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification.
Link: `https://github.com/yourh/AttentionXML`

[2] Ronghui You, Zihan Zhang, Suyang Dai, Shanfeng Zhu, HAXMLNet: Hierarchical Attention Network for Extreme Multi-Label Text Classification
Link: `https://arxiv.org/pdf/1904.12578.pdf`

[3] The Extreme Classification Repository: Multi-label Datasets & Code
Link: `http://manikvarma.org/downloads/XC/XMLRepository.html`