Multi-intent classifier based on a toxic comment dataset as archived in Kaggle (https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge).

In this task, we are provided with comments from wikipedia and each comment is labelled with six possible types of comment toxicity (*toxic, severe toxic, obscene, threat, insult, identity hate*). These are binary labels, but the submission is evaluated with the ROC AUC metric hence we need to predict the probability of belonging to each toxicity type. The task is hence a binary classification task with multiple labels for text (sequence) data.

Work done on this topic and experiments conducted are available at this repo: https://github.com/varshakr24/Kaggle_ToxicComment

Data Analysis and PreProcessing

https://github.com/varshakr24/Kaggle_ToxicComment/blob/main/notebooks/DataPreprocess.ipynb
Before building the model, text sequences were analysed to derive useful statistics required for modelling hyper parameters. Following are the few observations and decision made with respect to pre-processing of 'comment_text':

- The maximum length of each of the given training sequences was examined. It is clearly evident from the histogram that the majority of sentences are restricted in length to less than 100. Hence, a maximum length of 100 was chosen for sentence encoding in all experiments.
- Word frequency statistics were obtained and it was seen that words such as 'wikipedia', 'page', 'article' etc. tend to appear most number of times (Obvious given the source of comments). But, these words do not affect the toxicity of the sentence. Hence such words were filtered.
- Image tags e.g. Image:apple23.jpg frequently appeared in the comments. Again, this does not affect the toxicity value. Such tags were removed using regular expressions
- Punctuation, numbers etc - Anything other than words cannot be handled by our vectors. Such characters are removed. Newlines are handled.
- Emoji's are converted to regular texts as most of the time emoji tells a lot about the sentiment of the user and hence will give more information about the toxicity values.
  Ex: 👍 is converted into :thumbs_up:
- Spelling mistakes are quite common as they are user comments. A spell correction module could be used (Commented it out as it takes long time to run)

The cleaned files are available in the 'cleaned/' folder

Data modelling

Experimented with different models based on the word/sentence embeddings considered. The embeddings used are as follows. Code for running models corresponding to each embedding are in individual notebooks

- Glove: Trained on wiki corpus, hence source matches. Used spacy tokenizer. Restricting the maximum number of words in the vocabulary to 50000. Requires encoding architecture as Glove gives only word embeddings. Architectures include different combinations and ordering of CNNs and RNNs which act as sentence encoders on top of word embeddings. All models converged and took around 3-8 mins per epoch. The leaderboard scores ranged from 0.979 to 0.983. Vec used: *glove.840B.300d*
  https://github.com/varshakr24/Kaggle_ToxicComment/blob/main/notebooks/Glove.ipynb

- Bert: Entirely using the bert model hence bert embeddings. The bert model used is 'bert-base-uncased'. Tried lightweight different variants available e.g. distilbert. Started with finetuning the bert model, the model did not converge. Further, freezed the bert weights and trained the Dense layers only. Though it converged, did not give good results on the leaderboard. https://github.com/varshakr24/Kaggle_ToxicComment/blob/main/notebooks/Bert.ipynb

- Universal Sentence encoder: Extremely time consuming (3 hrs per epoch). Will update results to the github repo once the training completes. https://github.com/varshakr24/Kaggle_ToxicComment/blob/main/notebooks/USE.ipynb

Architectural decisions:

- Almost all lstms/grus are bidirectional with 128 hidden layers (unless specified explicitly in table)
- CNNs with 64 output channels and kernel size is 3 for most models (unless specified explicitly in table)
- Batch size of 128
- Regularization after embedding layer
- Early stopping was used to stop training and loss was evaluated on 10% of the train set used as validation data

These were kept constant and individual components were altered to better compare the models. All models are available in this file: https://github.com/varshakr24/Kaggle_ToxicComment/blob/main/notebooks/models/all_models.py

Results Evaluation

The table below gives the model, epoch at which the results were considered, validation loss and accuracy for 10% validation data and the leaderboard score on kaggle:

Model	**Batch Size**	**Pretrained Embedding**	**Early stopping Epoch**	**Validation Loss, acc**	**Kaggle Leaderboard Score**
CNN2d mith kernel sizes 3,5,7 and pooled	128	glove	5	0.0563, 0.9267	0.956
CNN followed by bi-LSTM of 64 units	128	glove	3	0.0439, 0.9868	0.98144
Only bi-LSTM	128	glove	3	0.0429, 0.9667	0.98257
bi-LSTM followed by 1 CNN1d	128	glove	2	0.0415, 0.9920	0.98316
bi-LSTM followed by 2 CNN1d layers and multiple dense layers	128	glove	4	0.0442, 0.9926	0.97936
bi-GRU followed by CNN1d	128	glove	2	0.0440, 0.9723	0.98251
bi-LSTM followed by 1 CNN1d - Unregularization	128	glove	2	0.0457, 0.9856	0.98218
BERT, Dense	64	bert	2	0.1453, 0.9934	-
Universal Sentence Encoder, Dense	128	USE	1	0.1041, 0.9709	-

The highlighted model gave the best score on the kaggle public leaderboard.

Key - Observations

- Simple model with 1 lstm layer, 1cnn layer and 1 dense layer itself performed best compared to more complex models like multiple cnns with different kernel sizes
- LSTM/GRU followed by CNN performed better than CNN followed by LSTM
- Complex models involving BERT or sentence encoders require a lot more training time and resource to achieve the score that simpler models do with lesser training time and resource.

Improvements

- Better language preprocessing; most comments include multilingual texts and all that data is lost.
- Punctuation is not handled. “!!!!!” can give a great amount of information
- Better hyperparameter tuning
- Ensembling multiple best performing models
- Efficient use of sentence encoding like USE.