Conquering Online Toxicity

Liam Ormiston
Patrick Canny
Jakob Wulf-Eck
Austin Adams

Problem

- Based on Kaggle's *Toxic Comment Classification* challenge
    - Part of an initiative by Alphabet's Conversation AI team to protect voices in online conversation
- Goal - to programmatically identify *toxic* comments
- Toxic is here defined as **'rude, disrespectful or otherwise likely to make someone leave a discussion'**

Dataset and Formatting

- Our dataset contains the text of approximately 2,000,000 online comments
- Provided by Civil Comments, a defunct commenting plugin for news sites
  - Significantly, encouraged users to vote on whether other users’ comments were civil or not.
- The original training set contained a toxicity score, as well as 5 other scores which rate comments on properties such as obscenity or threatening content.
  - Entrants asked only to predict the overall toxicity
- This year’s competition dataset adds additional scores indicating whether the comment mentions any of a variety of racial, religious, and gender identities, along with sexual orientations and disability statuses.

Our Approach: Methodology

- Calculate initial toxicity weights for each word based on a naive metric (appearances in toxic/good comments)
- Preprocess the data, truncating/padding all comments to a standard word length and representing words by their weights
- Instead of using a tokenized sequence of words, use this sequence of word *weights* for each comment as input to a neural network
- Using the neural network, predict the toxicity value of each comment

Our Approach: Tech

- Standard Python ML/data exploration stack
  - Jupyter
  - Pandas/Numpy
  - Matplotlib
- Keras for neural network implementation
- Simple neural network design
  - 200-neuron input layer (to process comment word-weight lists padded to 200 words)
  - Two 1/10 dropout layers to improve generality, interlaced with:
  - Two 50-neuron hidden layers
  - 1 output neuron for toxicity target score
    - Could be extended to predict additional training set parameters

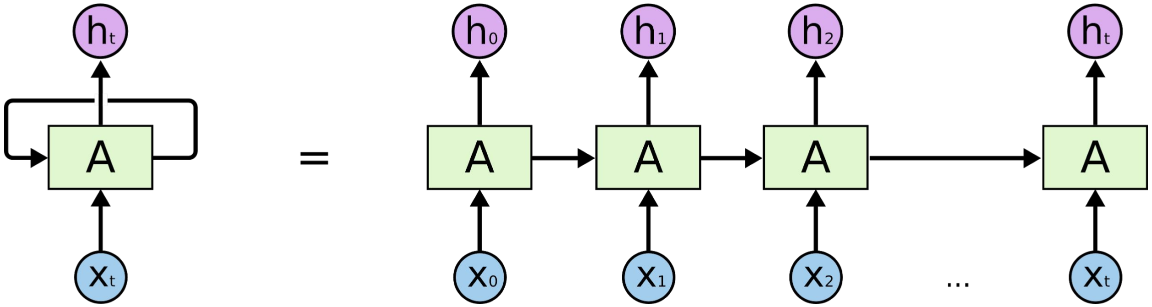
Our Results

- Binary cross-entropy accuracy is good- ~70% on the test set and ~69% on the validation set
- Higher predictions tend to map to higher toxicity, but predictions don't map well to real toxicity targets
- Word-weight method could likely be improved through use of a premade word sentiment analysis dataset

```
Train on 1624386 samples, validate on 180488 samples
Epoch 1/2
1624386/1624386 [==============================] - 112s 69us/step - loss: 0.3362 - acc: 0.7022 - val_loss: 0.3448 - val_acc: 0.
6879
Epoch 2/2
1624386/1624386 [==============================] - 115s 71us/step - loss: 0.3302 - acc: 0.7022 - val_loss: 0.3449 - val_acc: 0.
6879
```

Other Approaches

- In place of our word-weight method, we discovered that most highly-rated Kaggle submissions use word embedding
  - Representing words as vectors representing their relative ‘closeness’ in the data’s vocabulary in order to cluster toxic and non-toxic words
- Higher-dimensional nature of an embedding allows for the use of more specialized NN designs
  - LSTM (long short-term memory) - recurrent neural network layer
- These approaches allow for kernels which obtain ~90% accuracy on the test set.
- We implemented a more standard approach in a separate Jupyter notebook



Other Approaches: Methodology

- Tokenize the comments and pad to length 200
    - Map each word to an int, then represent every word by its int equivalent
- Pass to a NN using an embedding layer and an LSTM layer
    - Neural network learns a mapping of words to float vectors which places closely-related words together in vector space
    - LSTM has ‘memory’, learns patterns among long sequences of these vectors - like our comments
- Then run through a standard NN with 2 dropout layers and 2 dense layers to produce a result
- While a baseline version of this approach still produces values that don’t look much like the training labels, it more clearly differentiates toxic and non-toxic comments.

Conclusions

- Classifying comments by toxicity is a challenging problem
- Our word-weighting approach likely falls prey to a problem raised by the creators of the Kaggle competition:
    - How do we avoid bias - that is, models which associate non-toxic words which appear in toxic comments with toxicity?
    - For example, if 90% of comments containing the word ‘gay’ are toxic, how do we avoid flagging comments which use the word ‘gay’ positively?
- The LSTM approach is a good start to answering these questions, but there are improvements to be made
    - Precalculated embeddings, different neural net architectures, different metrics for success...

Conclusions

To find our repo, search ‘EECS738 Toxicity Classification’ on GitHub, or visit https://git.io/fjGzr.

Any questions?