Machine Learning Nanodegree
Toxic Comment Text Classification Challenge

Sam Mottahedi

01/08/2018

1 Definition

1.1 Project Overview

Natural language processing (NLP) is one of the most important technologies of the information age. Understanding complex language utterances is also a crucial part of artificial intelligence. Applications of NLP are everywhere because people communicate most everything in language: web search, advertisement, emails, customer service, language translation, radiology reports, etc. There are a large variety of underlying tasks and machine learning models behind NLP applications.

1.2 Problem Statement

Free expression and sharing information is the greatest impact of internet in modern society. Unfortunately, online abuse and harassment can lead limit self expression on the web. Many platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

In this Kaggle competition , participant are challenged to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity,insults, and identity-based hate.In order to accomplish this objective, a Recurrent Neural Network (RNN) Architecture with Gated Recurrent Units (GRU) were used which are suitable for processing information in the form of sequence of words. An Attention Mechanism is used in order to improve the performance of the model by focusing on important parts of input sequence. Multi-Task learning objective is added to network to improve the generalization error and reduce over-fitting to the training data set.

1.3 Metrics

Submissions are evaluated on the mean column-wise log loss. In other words, the score is the average of the log loss of each predicted column.

Since class labels are not mutually exclusive, a multi-label classification loss function is required here. Multi-label classification (MLC) is a prediction problem in which several class labels are assigned to single instances simultaneously as follows:

$$loss(\hat{y}, y) = \frac{1}{|L|} \sum_{l=1}^{l=|L|} -(y_l - log(\hat{y_l}) + (1 - y_l) \cdot log(1 - \hat{y_l})) \tag{1}$$

2 Analysis

2.1 Data Exploration

The Kaggle competition dataset provided is Wikipedia Human Annotations of Toxicity on Talk Pages and contains 160, 000 human labelled annotations based on asking 5000 crowd-workers to rate Wikipedia comments according to their toxicity (likely to make others leave the conversation). Each comment was rated by 10 crowd-workers. The Test dataset which is used to evaluating performance in competition consist of 226, 998 unlabeled comments. Each comment in the training set can be labeled with 6 labels which are not mutually exclusive.

The toxic comment data set have the following fields:

- ”id”: (string)
- ”comment texts”: comments (string)
- ”toxic”: toxic comment label (binary)
- ”sever toxic”: severely toxic comment label (binary)
- ”obscene”: obscene comment label (binary)
- ”threat”: threatening comment label (binary)
- ”insult”: insulting comment label (binary)
- ”identity hate”: identity hate comment label (binary)

2.2 Exploratory Visualization

It is important to know how labeled comment are distributed in the dataset and wether or not they the labels as balanced. Figure 1 shows the distribution of labels

The plot shows that most of the comments are innocent and small number of comments are classified as any form toxic comment. The comments classified as severely toxic, threat and identity hate are rare compared to toxic, obscene and insult.

An example of a comment is provided bellow with was labeled as both toxic and a threat.

```
('Hi! I am back again!\nLast warning!\nStop undoing my edits or die!',
```

As it can be seen, the comments are not clean and contain many signs, symbols and spelling errors that makes the classification problem based on word level vectors much harder.

It’s also helpful to look at word could of comments labeled with any of toxic comments labels (Figure 2):

The data will be divided in to 3:1:1 for training validation and testing. Since the labels are not balanced, during the training each batch of data will be sampled using a Stratified sampling method to expose the model to different class labels.

2.3 Algorithms and Techniques

Recurrent neural network architectures combining with attention mechanism, or neural attention model, have shown promising performance recently for the tasks including speech recognition, image caption generation, visual question answering and machine translation. In the sequence labeling tasks, the model input is a sequence, and the output is the label of the input sequence. The major difficulty of sequence labeling is that when the input sequence is long, it can include many noisy or irrelevant parts. If the information in the whole sequence is treated equally, the noisy or irrelevant part may degrade the classification performance. The

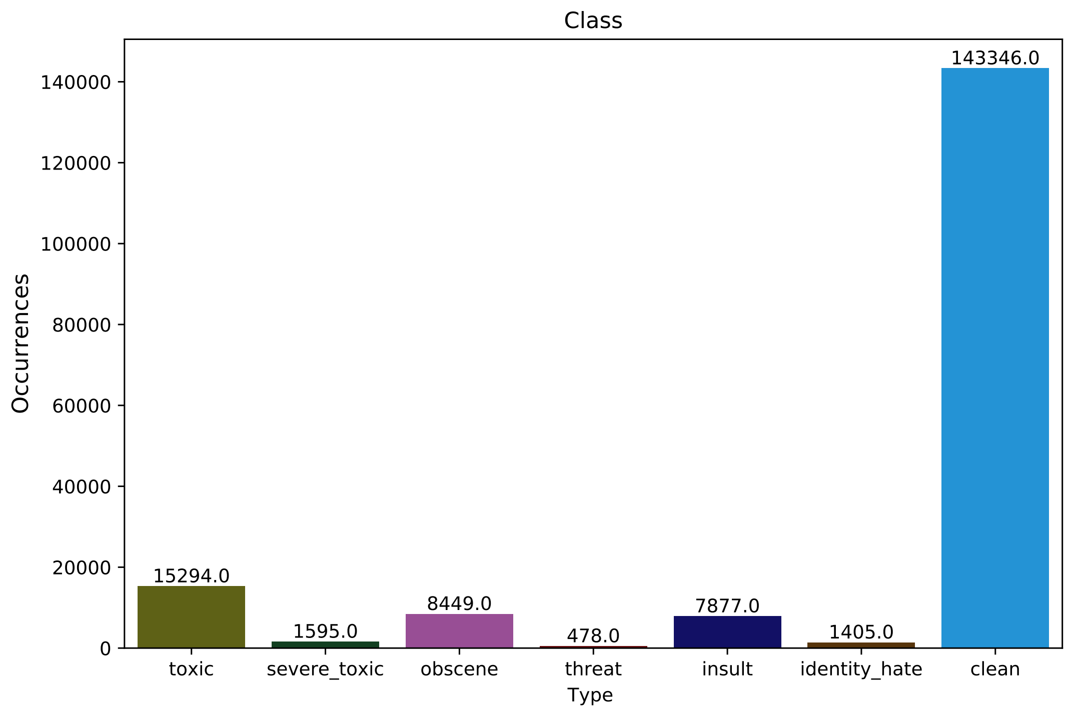

Figure 1: Toxic comment dataset label distribution

attention mechanism is helpful for sequence classification task because it is capable of highlighting important part among the entire sequence for the classification task.

Recurrent neural networks (RNNs) are class of neural networks designed for processing sequential data [Goodfellow et al. (2016)]. Compared to feed forward neural networks, RNNs are able to scale to longer sequences which in most part is not practical without sequence-based networks. Generally, RNN are implemented by unrolling the computational graph and sharing wmodel parameters across deep network structure. RNNs are trained using Back-Propagation Through Time (BPTT) which is variation of ordinary back-propagation applied to an unrolled computational graph. Deep RNNs usually face complication due to Long-Term dependencies [Goodfellow et al. (2016)] such as vanishing and exploding gradients. Most effective solution to overcome these problem is the use of Gated Recurrent Units (GRUs) which is based on idea of creating pathways through time that have derivatives that neither vanish nor explode. GRUs accomplish this by introducing self loops to add pathways were gradient can flow for long duration.

In this project, each comment is treated as set of features using pre-trained GloVe word embedding. A Bi-directional recurrent neural network with Gated Recurrent Units (GRU). Attention mechanism is added on top of the Bi-directional RNN which improve the performance of the model focusing on important parts of long sequence and reduce the effect of noise and unrelated information. The classification task is a multi-label classification where comments toxicity labels are not mutually exclusive.

2.4 Benchmark

The baseline chosen here in order to get better understanding of the problem from a general machine-learning perspective. To this end, the baseline model is chosen to be a logistic regression model based on Term Frequency-inverse document frequency data which can achieve column-wise log-loss of 0.05567.

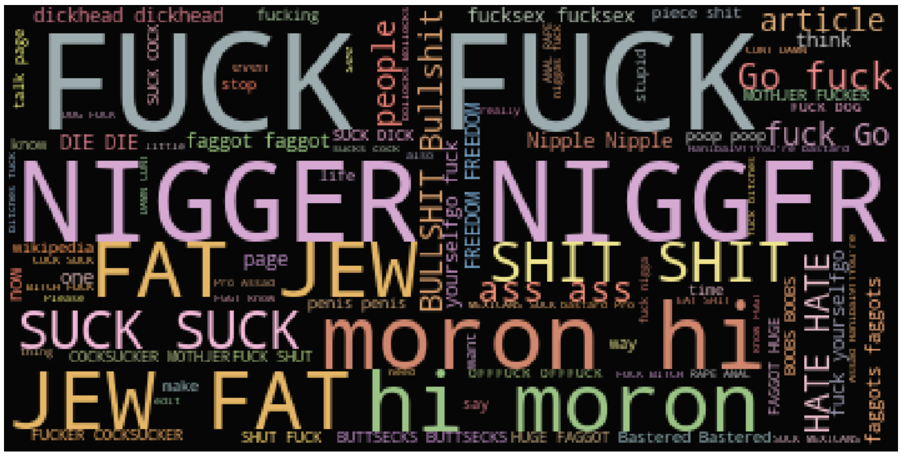

Figure 2: Toxic Comment WordCloud

3 Methodology

3.1 Data Processing

The preprocessing is mostly done using functions available in data.py module. The preprocessing step before training or testing consist of following steps:

- Preprocessing comments:
  - tokenizing
  - converting to lower-case letters
  - removing stopwords
  - normalizing numbers, date, ...
- Vocabulary: creating dictionary of words with at least $count() > Thresh_hold$.
- Index to word: dictionary with keys equal to word index and values being a word in the vocabulary
- Word to index: dictionary with keys equal to vocabulary's word and value being words index.
- Sequence index file: a file containing sequence of word index for each comment.

During training, the pre-processed comment are read from sequence to index file and push to TensorFlow input placeholders. In addition, at the beginning of the training the word vectors corresponding to the words in the vocabulary are extracted from pre-trained GloVE word vector with 6 Billion words and specified word vector dimension.

The preprocessing process has the following adjustable parameters:

- Word frequency threshold
- GLoVe word vector dimension (50, 100, 200)
- Using trainable or not trainable word vector tensor
- Maximum input sequence length

3.2 Implementation

3.2.1 Text Classification with Recurrent Neural Networks

The architecure used in this work is Recurrent Neural Network (RRN) that takes the sequence of words index and an encode the information in the text in the last output layer. First we need to lookup word-vectors corresponding to each word in the input sequence. The embedding matrix and the embedding lookup operation defined using TensorFlow as:

```
@lazy_property
def _create_embedding(self):
    with tf.name_scope('Embeddings'):
        if self.pre_train:
            self.embedding = tf.Variable(tf.constant(0.0,
                                                    shape=[config.VOCAB_SIZE,
                                                    config.GLOVE_SIZE]),
                                                    trainable=config.TRAINABLE_EMBEDDING)
            self.embedding_init = self.embedding.assign(self._embedding_placeholder)

        else:
            self.embedding = tf.Variable(tf.random_uniform(
                [config.VOCAB_SIZE, config.EMBEDDING_DIMENSION], -1.0, 1.0))
embed = tf.nn.embedding_lookup(self.embedding, self._inputs)
```

Since some of the comments are quite long, a Bi-directional RNN is used which has two Gated Recurrent Cells (GRU) for forward and backward processing of the input sequence and outputs the concatenated output oof the forward and backward cell [figure 3]. The GRU forward and backward cell are and the bi-directional dynamic RNN are defined in the `_inference` method of `SeqClassifier` class:

```
with tf.name_scope('Bi-GRU'):
with tf.variable_scope('forward'):
    gru_fw_cell = tf.contrib.rnn.GRUCell(
        config.HIDDEN_LAYER_SIZE)
    if self.mode == 'train':
        gru_fw_cell = tf.contrib.rnn.DropoutWrapper(gru_fw_cell,
        output_keep_prob=config.KEEP_PROB )
with tf.variable_scope('backward'):
    gru_bw_cell = tf.contrib.rnn.GRUCell(
        config.HIDDEN_LAYER_SIZE)
    if self.mode == 'train':
        gru_bw_cell = tf.contrib.rnn.DropoutWrapper(gru_bw_cell,
        output_keep_prob=config.KEEP_PROB)

    outputs, states = tf.nn.bidirectional_dynamic_rnn(cell_fw=gru_fw_cell,
                                                cell_bw=gru_bw_cell,
                                                inputs=embed,
                                                sequence_length=self._seq_length,
                                                dtype=tf.float32,
                                                scope='Bi-GRU')
states = tf.concat(values=states, axis=1)
```

It is important to note that the Dropout should only apply at the time of training. The last output layer of the RNN is concatenated and passed through two feed forward layers and a final layer with 6 output nodes where a element wise sigmoid function is applied and a multi-label predictions are generated.

The optimizer used here is Stochastic Gradient Decent with exponentially decaying learning rate defined as:

```
    def _create_optimizer(self):
    with tf.variable_scope('training') as scope:
        self.global_step = tf.Variable(0, dtype=tf.int32, trainable=False,
                                    name='global_step')

        if self.mode == 'train':
            with tf.name_scope('learning_rate'):
                self.learning_rate = tf.train.exponential_decay(config.LR, self.global_step,
                                    10000, 0.96, staircase=True)
                tf.summary.scalar('learning_rate', self.learning_rate)
            self.optimizer = tf.train.GradientDescentOptimizer(self.learning_rate)
            trainbales = tf.trainable_variables()
            start = time.time()
            clipped_grads, self.gradient_norms = tf.clip_by_global_norm(tf.gradients(self.losses,
            trainbales), config.MAX_GRAD_NORM)
            self.train_ops = self.optimizer.apply_gradients(zip(clipped_grads, trainbales),
            global_step=self.global_step)
            print('creating opt took {} seconds'.format(time.time() - start)
```

It can be seen that the gradient are manually applied and the gradient norm is clipped at 1 to prevent problems due to exploding gradients.

3.3 Refinement

3.3.1 Attention Mechanism

Since not all words contribute equally to the representation of the sequence meaning we need to use an attention mechanism [Yang et al. (2016)] that extract important words to the meaning of the sequence and aggregates the presentation of informative words to form a sentence vector from [figure 4

$$u_{it} = tanh(W_w h_{it} + b_w) \tag{2}$$

$$\alpha_{it} = \frac{exp(u_{it}^T u_w)}{\sum_t exp(u_{it}^T u_w)} \tag{3}$$

$$o = \sum_t \alpha_{it} h_{it} \tag{4}$$

where the word annotation h_{it} is passed to a one-layer MLP to get u_it as hidden representation h_it the importance of the word is measured by comparing u_{it} to a word context vector u_w and then passed through a softmax function to get normalized importance weight α_{it} and the output vector is weighted sum of word annotation based of the weights. The context vector is randomly initialized and jointly learned during the training process.

The completed TensorFlow computational graph can be seen in Figure [5]

3.3.2 Multi-Tasak Learning

In order to prevent over-fitting to training set and auxiliary loss was added to the computational graph. Caruana (1998) showed that multi-task learning improves generalization by leveraging the domain-specific information contained in the training signal of related task. Here, the auxiliary task used is the log-loss error for predicting if any form of comment toxicity is observed. Since large number of comments didn't classified as toxic comment, this auxiliary task help better discriminating toxic comments. The scheme used here is Hard Parameter Sharing Caruana, which the parameters are shared between all task while keeping several task-specific output layers (Figure 6).

3.3.3 Other Measures

In addition to the attention layer other measures such as dropout layers, exponential learning rate decay were used to reduce the log-loss and improve the model generalization.

4 Results

4.1 Model Evaluation and Validation

During development process, the training data was split into training (90%) and validation (10%) sets. Since this project is part of Kaggle competition, instead of using a test set the submission result is used as a metric for final evaluation of the model.

The final description of the final model is as follows:

- Batch Size = 256
- Embedding dimension = 100
- Number of GRU Cells = 128
- Feed-forward layers = 128

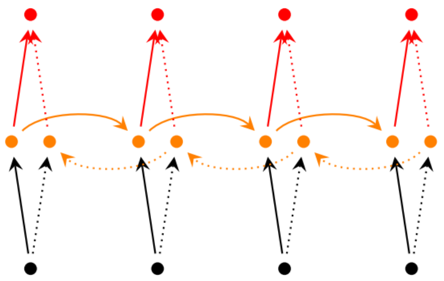

Figure 3: Bi-directional RNNs

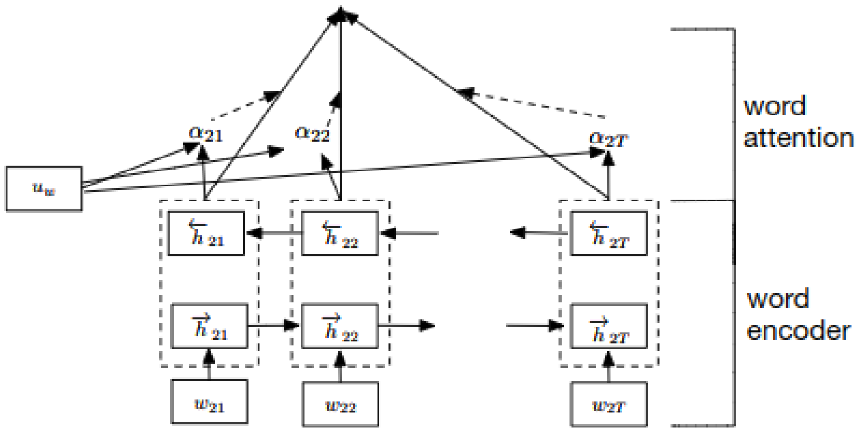

Figure 4: Attention Mechanism Yang et al. (2016)

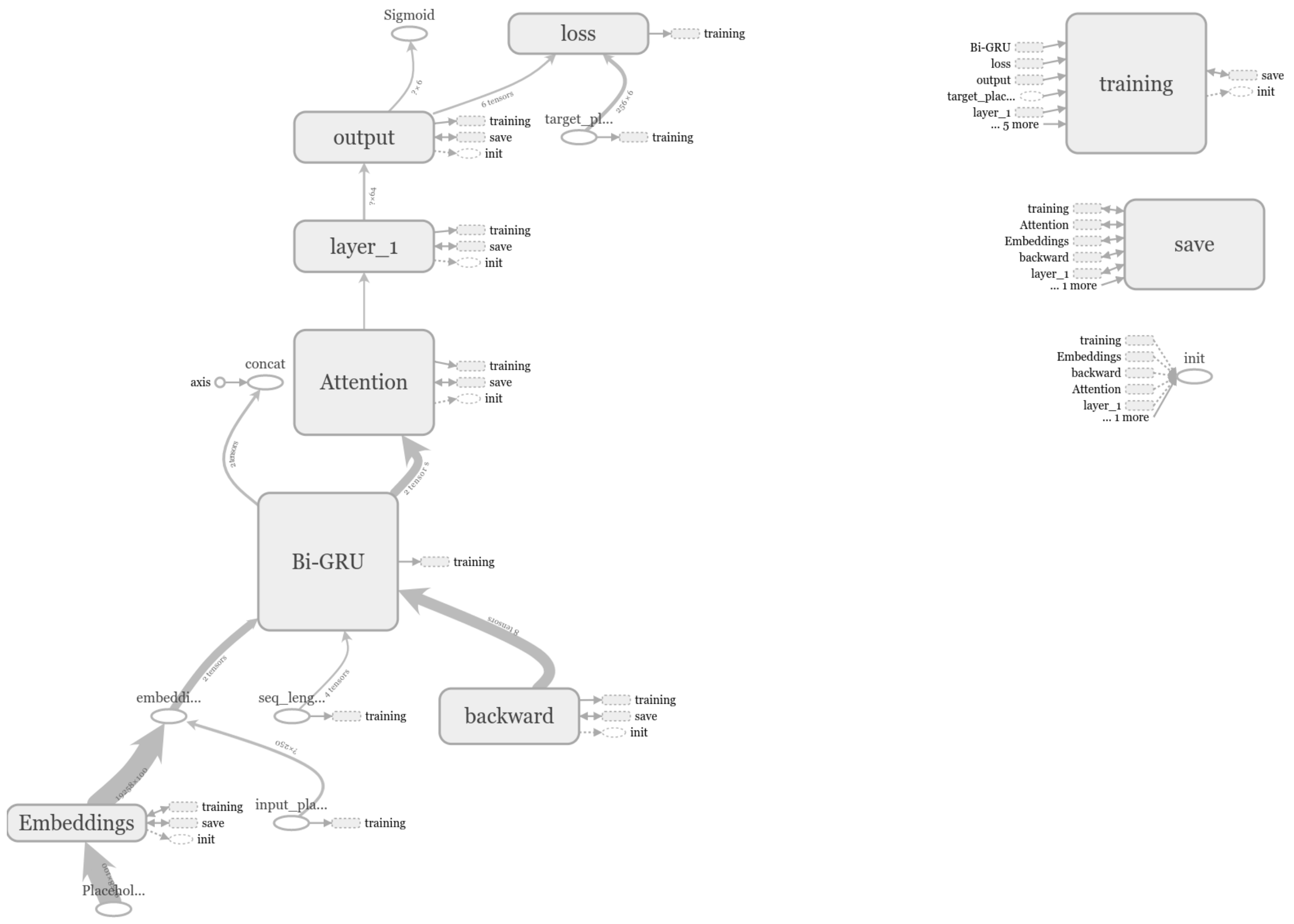

Figure 5: TensorFlow computational graph for sequence classification model with attention mechanism

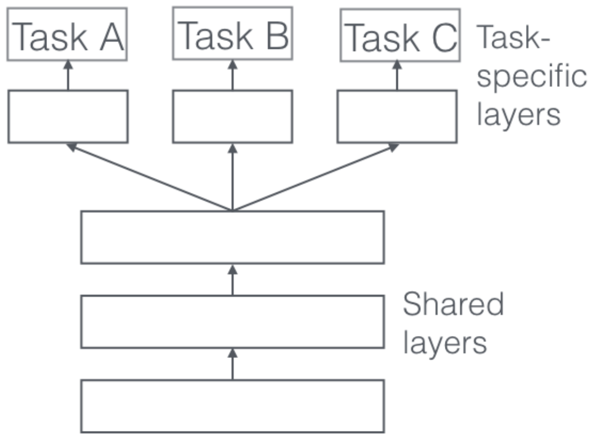

Figure 6: Multi-task learnng with Hard Parameter Sharing.

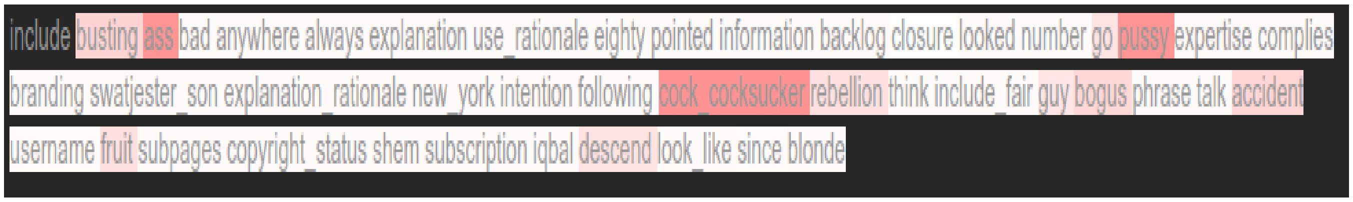
include busting ass bad anywhere always explanation use_rationale eighty pointed information backlog closure looked number go pussy expertise complies
branding swatjester_son explanation_rationale new_york intention following cock_cocksucker rebellion think include_fair guy bogus phrase talk accident
username fruit subpages copyright_status shem subscription iqbal descend look_like since blonde

Figure 7: Attention layer Visualization

- Attention size = 256
- Maximum input sequence length = 250

An early-stopping mechanism is implemented to automatically save model weights corresponding to best evaluation log-loss and stop the training to prevent over-fitting to the training set.

4.2 Justification

Using the RNN based text classification with attention mechanism and the lowest log-loss error achieved is 0.0342 which shows a 38.56% reduction compared to log-loss error obtained using the benchmark model. The reduction in error is significant and justifies the computational cost of training a deep architecture.

5 Conclusion

5.1 Free-Form Visualization

In order to visual the significance of solution used for this specific problem, attention layer weights and the increase importance of words in the input with higher attention weights is provided in Figure 7. It can be seen that model learned to pay more attention to words that contain profanities.

5.2 Reflection

In this project a deep recurrent neural network was trained to predict if the input text includes multiple profanity or contains content that and can negatively effect the online community. Since the text in the comments include many non-standard form form of speech or incorrect spellings, significant effort was required for preprocessing the text. Since in most cases, the observed comments were long, a recurrent neural neural network with attention mechanics was implemented that is immune to common problem in these situation such as vanishing gradient, exploding gradient and etc. The proposed model was compared to a logistic1 regression model based on tf-idf data which shows significant reduction in log-loss error and justifies the additional computational cost.

5.3 Improvement

This project can use improvements in many respects, First, larger model with many additional hidden layers can improve the generalization and result in better prediction. Also, more training data can certainly improve the results. Other measures such as pre-training on similar data set can alo improve the generalization. In addition, like many Kaggle competition the best results are based on ensemble of many models and is possible solution to improve the predictions.

References

R Caruana. Multitask learning: A knowledge-based source of inductive bias1. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Citeseer.

Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.