CS249 Project Report: Quora Insincere Question Classification using Context-Dependent Embeddings

Yingbo (Max) Wang
604593537
yingbowang@ucla.edu

Christian Warloe
004584729
cwarloe@ucla.edu

Yinxue Xiao
904581627
yolandaxiao7@gmail.com

Wenlong Xiong
204407085
wenlongx@gmail.com

1. Introduction

Quora is a popular online platform self described as ”A place to share knowledge and better understand the world” that hosts online discussions through user submitted questions and answers. Used properly, the website can be a useful tool, but the open nature of the website and question and response submission process puts Quora at risk from bad actors who may attempt to derail discussion and create toxicity.

There are two aspects to this problem: identifying comments and questions that are meant to derail discussion. Some prior work has been done on the identification of toxic comments [1][2], but the process for identifying toxic questions remains relatively unexplored. Currently, Quora’s process to identify these bad actors relies primarily on human moderators who manually police content on the site. However, the large amount of content generated on the website greatly outstrips the amount of moderators Quora employs, which motivates the need for a way to automatically identify toxic of insincere content, reducing the human effort needed for moderation.

Most of the recent work in creating classifiers for identifying if a Quora question submission is valid has been done as part of the Quora Insincere Question Kaggle competition[3]. These classifiers detect if a question is sincere and insincere, where the ”insincere” tag is a conservative way of flagging questions that contain toxic content. As defined by Quora, insincere questions can be divided into four categories: questions with a non-neutral tone, questions that are disparaging or inflammatory, questions that are not grounded in reality, and questions that use sexual contents for shock value. These classifiers submitted to the Kaggle competition represent the current ”state of the art” work, and rely heavily on recurrent neural network (RNN) architectures, such long short term memory (LSTM) units or gated recurrent units (GRUs). However, these recurrent networks require a large amount of computational resources and a long training time, and are not ideal for applications such as Quora’s that deal with a high volume of examples. As a result, the goal of our project is to achieve high performance (similar to state-of-the-art methods) using simpler model architectures that are faster to train.

In order to simplify the classifier architecture while maintaining similar performance, we leverage recent advances in word embedding models. Prior state-of-the-art methods use context-independent word embedding methods to generate vector inputs to the classifiers; this necessitates recurrent structures in order to capture dependencies across the sentence. However, recent word embedding models that utilize contextual information are able to directly capture these dependencies, and incorporate them into the word embeddings. By incorporating contextual information into the word embeddings themselves, we hope to eliminate the need for recurrent structures. We explore using two of these recent embeddings, ELMo[4] and BERT[5] in conjunction with simpler classifiers (logistic regression, dense neural networks, and convolutional neural networks), in order to obtain similar performance to the state-of-the-art models.

2. Related Work

2.1. Toxic Comment Classification

The related problem of toxic comment classification has been explored by some previous works. Like detecting insincere questions, toxic comment classification is a form of sentiment and semantic analysis. Prior work includes that done by Georgakopoulos et al. [1], who used a convolutional neural network on word embeddings generated using word2vec to classify toxic comments from Wikipedia dataset. He compares CNN performance against bag-of-words models, and demonstrates that structure is important to toxic comment classification. Aken et al. [2] evaluated various classifiers for toxic comments based on Wikipedia and Twitter datasets as well. They compared logistic regression, CNN, and RNN models, and demonstrated that recurrent models (more specifically, bidirectional gated recurrent units / GRU with an attention layer on word vectors using GloVe and FastText embeddings) performs the best for individual classifiers, and utilizing ensemble learning outperforms all individual classifiers.

These related works demonstrated that more structured deep learning models tended to result in better performance on toxic text classification, with recurrent networks gaining the best performance at the cost of longer execution time. In particular, Aken et al. [2] influenced our decision to use Logistic Regression and Convolutional Neural Network models as the "simpler" classifiers in our project, since these two model types were shown to achieve good yet lesser performance when compared to recurrent neural networks.

2.2. Quora Insincere Question Classification Kaggle Competition

All of the prior work done for this specific Quora question classification task are the models submitted to Quora Insincere Question Classification Kaggle competition[3]. In this specific competition, four types of pretrained word embeddings were provided for the training and test data; these included embeddings using the word2vec, GloVe, Paragram, and FastText models. There were over 1,400 unique models submitted to this competition, and the vast majority of them used a recurrent neural network architecture in conjunction with one or more of these pretrained word embeddings to create their classifiers. In addition, due to imbalances in the dataset classes for this specific competition, classifiers are evaluated using F1 score.

In particular, the top performing model is used as the baseline "state-of-the-art" model that we compare our models against. It utilized the GloVe and Paragram word embeddings, and used a bidirectional LSTM, CNN, and dense neural network together to create their model. This top performing model achieved an F1 score of 0.71323, and also utilized additional statistical features from the raw question text, such as capitalization or special characters. The specific architecture will be described more in detail in later sections.

3. Classifier Models

In the following section, we describe some of the model types in prior work and our approach, as well as the intuition for why they work well on this specific dataset.

3.1. Logistic Regression

Logistic regression is a basic linear classifier used for binary classification. The input to logistic regression is a vector, and the output is a scalar value between 0 and 1 (commonly thresholded at 0.5 to convert the scalar output to a categorical output). The input vector to the logistic regression model is linearly transformed, then passed through a sigmoid function to rescale the output to the $[0, 1]$ range. The model can be represented by the equation:

$$y = \frac{1}{1 + e^{-(WX+b)}}$$

where X is an input vector, W is a vector of weights, and b is a bias term. Logistic regression is simple and quick to apply to a dataset, but due to its simple linear nature, can only obtain perfect accuracy if the dataset is linearly separable. In the case where data is not linearly separable, more complicated models are necessary to obtain high classification accuracy.

3.2. Neural Networks

Deep neural networks are relatively recent models that consist of multiple layers of perceptrons/neurons, that take a tensor or vector input and generates a corresponding tensor output, depending on the architecture. Each neuron takes a tensor input and applies a linear transformation, then applies a nonlinear thresholding function (commonly called a nonlinearity). This allows deep neural networks to approximate any nonlinear function[6], making them more expressive than simple linear models like logistic regression.

3.2.1. Dense Neural Networks

Dense neural networks consist of fully connected layers, where every neuron in a given layer is connected to all the neurons of the previous layer. Dense neural networks are the most simple neural network architecture, but have been shown to obtain good performance on a variety of tasks due to their ability to model nonlinear functions.

3.2.2. Convolutional Neural Networks

Convolutional neural networks are a special class of neural networks, that utilize convolutional layers. Convolutional layers differ from dense layers in that they have convolutional kernels; a small set of neurons that is spatially shifted (convolved) over neurons of the previous layer. In comparison to fully connected layers, convolutional layers have much fewer parameters and are therefore faster and easier to train. In addition, the fact that kernels are convolved means that spatial relationships are better represented with CNNs than dense neural networks. Empirically, CNNs perform better than dense neural networks, and have been shown to be effective at classifying toxic text.

3.3. Recurrent Neural Networks

Recurrent neural networks are a class of neural models that take in a sequence of tensors, and generate an output sequence of tensors. Vanilla RNNs examine the input tensors one by one, and at each time step, generate an output tensor while ”remembering” information about the input in a hidden state that it passes on to the next time step. Because of this, RNNs are considered the ideal model for analyzing text, since sentences are just a sequence of words that are interrelated. RNNs can capture relationships between words in a sentence, and generate output tensors that represent these relationships. However, due to its inherently sequential nature, RNNs are forced to process each element of their input sequence individually and are difficult to parallelize. Because of this, RNNs are generally slower and require more computational resources during training and prediction than similar size CNN models.

3.3.1. LSTM and GRU

In theory, RNNs are able to capture dependencies between individual tensors in the sequence of inputs, but in practice, have difficulty doing so. Variations on the vanilla recurrent neural network model have been introduced, that modify the process of ”remembering” and storing information in the hidden state, to better learn longer-term dependencies. Two of these variations are called Long Short Term Memory (LSTM), and Gated Recurrent Unit (GRU). Both LSTM and GRU achieve better performance than vanilla RNN on tasks that act on sequential data, while being faster to train. The performance differences between LSTM and GRU are minor however.

3.3.2. Bidirectional RNNs

Because vanilla RNNs process its sequence of inputs in order, it is better at modeling dependencies from later inputs to earlier inputs than the other way around. Bidirection RNNs solve this issue by essentially combining the outputs from two RNNs; in one RNN, the input sequence is processed in normal order, while in the second RNN (that has the same architecture), the input sequence is processed in reverse order. The outputs corresponding to the same input values of both RNNs are concatenated, so that each combined output contains information about dependencies to both elements before and after that given input element.

4. Word Embeddings

Word vectors are commonly used as inputs to classification models instead of raw text, because they can capture semantics and relationships between words. In this work, we examine three types of word embeddings.

4.1. GloVe

The first word embedding we use for our project is Global Vector (GloVe) embedding. GloVe word embedding vectors were proposed by Pennington et al.[7]

as a method for producing vector encodings for words. While previous works noted the existence of semantic meanings behind the relative distance and cosine similarity between words in the embedding space as a side effect of the training objectives used to obtain the word vectors, GloVe seeks to explicitly design these regularities with the training objective and evaluation methods.

GloVe trains word vectors to encode the global co-occurrence probabilities of words within a certain corpus, which Pennington et al shows to be similar to previous methods such as that used by the skip-gram model [7]. By utiliizing global co-occurrence statistics instead of local ones, GloVe manages to encode word semantics into the vector embeddings, but at the same time is unable to encode information from a local context into the word vectors.

4.2. ELMo

The second word embedding we use for our project is ELMo[4]. ELMo is a deep contextualized word representation that models both word syntax and semantics, as well as how these across different contexts. ELMo is different from prior word embedding models in that it attempts to capture contexual information in the word vector, which means vectors could differ from sentence to sentence depending on the context in which it is used. It is also unique in that it represents a form of transfer learning. Pre-trained language models have been shown to improve the performance of many natural language processing tasks (Dai and Le, 2015; Peters et al., 2017, 2018; Radford et al., 2018; Howard and Ruder, 2018), so we expect to benefit from this architecture.

The ELMo model works by pretraining a language model, and using the hidden states from that language model in combination with the input word vectors to generate contextual word vectors. In the original paper, the ELMo model consists of 2 parts: a language model that is made up of 2 stacked bidirectional LSTMs, and a character-level embedding model that generates context-free word vectors from a sequence of characters. This character-level embedding model first takes in a raw sequence of characters and creates character embeddings, then passes them through convolutional and max-pool layers as well as a highway network to generate a context-free word embedding. Then, these word embeddings are used as input to the ELMO language model, which is trained on the 1B Word Benchmark dataset. This trained language model can then be used to transfer its learned results to the task of generating contextual word embeddings instead. The way that ELMo does this is that it concatenates three different vectors; the context-free word embedding the language model takes as input, the hidden layer in the first bidirectional LSTM, and the hidden layer in the second bidirectional LSTM. Since the language model's LSTMs were already pretrained, the two LSTM's hidden states capture contextual information from before and after the word in question, that is useful for predicting the next word. This captured information transfers over to well to the task of generating contextualized word embeddings, which is why concatenating the hidden states to the non-contextual embedding allow it to become a contextualized embedding.

Another benefit of the ELMo model is that, in addition to capturing contextual information, it also captures statistical information about the words itself, since it operates at a character level. Because ELMo initially generates character embeddings, it retains information about word capitalization, punctuation, spelling, etc and does not have any out-of-vocabulary words. This also gives us the added benefit of not having to do preprocessing on input text when generating ELMo embeddings.

4.3. BERT

The third word embedding we use for our project is BERT[5], which stands for Bidirectional Encoder Representations from Transformers. Like ELMo, it is a form of transfer learning, that uses pretraining on another NLP task to learn to generate good word embeddings. The BERT model consists of a multi-layer bidirectional Transformer encoder, which is identical to the original Transformer architecture proposed in Vaswani et al. [8], and takes a sequence of word tokens as input and generates an output of contextualized token embeddings as well as a "sentence" ([CLS] token) embedding. The embedding vector of each token is the summation of the following three vectors: a token embedding that encodes the word itself, a segment embedding that marks which sentence the word belongs to in a sentence-pair, and a position embedding that represents the word's context. Therefore, the embeddings of the same word would be different in the case when it has different meanings in multiple contexts (such as the word ”bank” in ”river bank” and ”bank account”).

BERT works by pretraining its bidirectional Transformer encoder on two separate tasks; one is an unsupervised task called Masked Language Modeling, and another is a supervised classification task called Next Sentence Prediction. In the unsupervised task, 15% of all input tokens are randomly masked and replaced by a single token ”[Mask]” in each sequence; the model predicts the masked word based on its contexts. In the second task, two sentences are picked from the corpus where one immediately follows the other 50% of the time and are random sentences from the corpus otherwise; the model predicts if the sentences are adjacent, using the outputted sentence embedding (corresponding to the input [CLS] token). BERT also supports fine-tuning, which is additional training of a pretrained model, but on a task-specific corpus to ”tune” the model to a specific task.

Like ELMo, BERT incorporates information from the left and right contexts (adjacent words in the input sentence) when generating word/token embeddings. This contextual information has been shown to be crucial for many NLP tasks, including question-answering related tasks. BERT differs from ELMo however, in that it uses Transformers instead of recurrent networks to extract relevant information from left and right contexts, which has been shown to be more effective than standard recurrent networks for encoding word embeddings[8]. BERT also generates embeddings at a token level instead of at a word level like ELMo does. What this means is that a sentence is first passed through a tokenizer to break words down into smaller units; for example ”running” would be broken down into the verb token ”run” and the present participle suffix token ”#ing”.

5. Current State-of-the-Art

Currently, we define the state-of-the-art model for this Quora Insincere Question Classification task as the top scoring model from the Kaggle competition. The model architecture is described above in Figure 1.

The model uses a weighted combination of two non-contextual word embeddings, GloVe and Paragram, then passes these embeddings through a bidirectional LSTM and a 1D convolutional network with dropout to generate an intermediate feature vector. In parallel, ”statistical features” are generated from the sentence; these statistical features are indicators or counts for features such as length of the text, number of capital letters, punctuation, number of special symbols, and number of unique words, etc. These statistical features were passed into a dense neural network, and output an intermediate feature vector as well. These two intermediate vectors were concatenated together, and then passed through a fully connected neural network, which output a binary classification. The model was trained using cross validation, with a hyperparameter search for the optimal classification threshold applied to each fold. The classifier also incorporated the idea of model ensemble, and combined the results of multiple instances of this model for its final prediction.

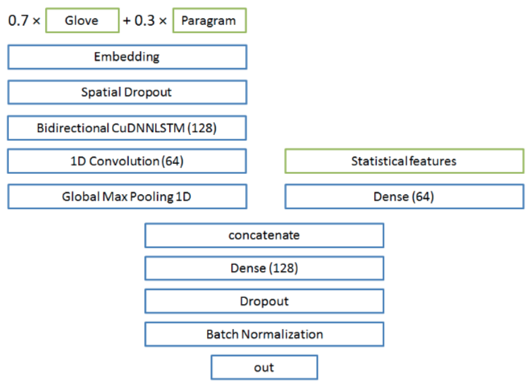

Figure 1: Model architecture for current state-of-the-art insincere question classifier. Figure reproduced from [3].

From this model we can see that the state-of-the-art incorporates several ideas. First, the use of a bidirectional LSTM almost directly on the word embeddings implies that recurrent networks are the most effective at analyzing information from the contexts words are used in, and for generating an intermediate representation that ”summarizes” a sentence into a vector. Second, the extensive use of convolutional and dense layers later on in the classifier imply that these simpler neural network layers are sufficient to generate the actual classification labels, once the LSTM has incorporated contextual information into the intermediate output, and that further recurrent structures are not necessary. Third, the fact that statistical features are used as an additional feature input implies its importance to this specific classification task. As a result, we consider each of these ideas when selecting our proposed models and embeddings.

6. Proposed Models

In our project, we attempt to achieve similar performance to current state-of-the-art models on the Quora Insincere Question Classification task, by using simpler models in conjunction with contexualized word embeddings. In this section, we explained the embeddings and models used, the reasoning behind choosing them, as well as any preprocessing performed.

6.1. Embeddings Used

6.1.1. GLOVE

We chose to use this non-contexualized word embedding because a large number of the top submissions to the Quora Insincere Question Classification Kaggle competition used GloVe word embeddings as inputs to their model. In addition, prior work such as that of Aken[2] demonstrated that it was one of the word embeddings that resulted in better performance (as compared to word2vec and FastText). Using this non-contextualized word embedding provides us a baseline to better quantify the improvement contextualized word embeddings provide vs the choice of classifier models themselves.

For this work, we used the pre-trained GloVe embeddings provided by Kaggle. The pretrained GloVe embeddings come from the model released with the original paper[7], which is trained on the Common Crawl dataset, and contains a vocabulary of 2.2 million words, and generates word embeddings of dimension 600. To generate words embeddings for each sentence in our training and test data, we perform some initial preprocessing: the sentences are split into word tokens, stripped of punctuation, and converted to lowercase. Then, each word is then converted to its corresponding embedding. Words without GloVe embeddings were ignored (skipped) for the sake of training and classification.

After preprocessing, we found that more than 99.5% of the words (as well oas more than 97% of unique words) in the training corpus contained GloVe embeddings. Because of this high coverage not attempts were made to specially handle out of dictionary words.

6.1.2. ELMo

We chose to use this contextualized word embedding because of 2 main reasons. The first is that it explicitly utilizes information from the left and right contexts in a sentence to generate word embeddings, and it does so by using a bidirectional LSTM. Since ELMo and the state-of-the-art classifier utilize similar recurrent architectures, we believe that ELMo can capture all the information that the state-of-the-art classifer uses its bidirectional CuDNNLSTM to capture. The second main reason that we chose ELMo is because it uses characters as its input instead of words. Because ELMo uses characters, we can embed the raw question text without preprocessing, and not have any out-of-vocabulary words. This allows us to capture information that would normally be removed during preprocessing, such as capitalization, punctuation, emojis, special characters; which correspond to the statistical features that are used in the state-of-the-art model.

The pretrained ELMo model that we used was released with the original paper in the library AllenNLP[4], and was pretrained on the 1B Word Benchmark dataset. Since ELMo operates on a character level which allows it to create an embedding for any word, we do not preprocess the raw question text at all, and allow ELMo to create word embeddings directly. This pretrained ELMo model creates a sequence of word embeddings of dimension 1024, from each raw question.

6.1.3. BERT

We chose to use this contextualized token embedding because of similar reasons to ELMo. BERT also explicitly utilizes information from the left and right contexts in a sentence to generate word embeddings. BERT uses the Transformer architecture, which was not very widely used in prior work for the Quora Insincere Question Classification task, but has been empirically shown to be better at capturing longer-term dependencies than LSTMs. We also chose to use BERT because it incorporates classification explicitly into its training process, by training a classifier on the ”sentence” embedding ([CLS] token) that it generates. We believe that training our models directly on this sentence embedding (in addition to the standard token embeddings) could also yield good results.

In this paper, we use the pre-trained uncased BERT-Base model that was released with the original paper[5]. This model uses 12 layers (Transformer blocks), 768 hidden states, and 12 self-attention heads to generate embeddings for each Quora question. The model is pre-trained on the concatenation of BooksCorpus with 800 million words and the English Wikipedia with 2,500 million words. BERT supports additional fine-tuning of the pretrained model, but this was not performed due to time and computing limitations of our project. We use the default BERT tokenizer and preprocessor to preprocess the question text, and generate a embedding of dimension 768 for each word token, as well as a an embedding of dimension 768 for the ”sentence” embedding / [CLS] token. The maximum sentence length is set to 32, due to memory constraints, but this is acceptable as no more than 5% of sentences in the dataset contain more than 32 tokens. Sentences exceeding this limit are truncated to 32 tokens. These embeddings are generated on a Dell Optiplex workstation with an 3.10GHz Intel Core i5-2400 CPU with 4 cores and 8GB RAM.

6.2. Classifiers

6.3. Input Processing

In order to convert variable length input data (sequences of word vectors corresponding to a question sentence) into the fixed length input data required by our models, we performed several preprocessing tasks.

For the fully connected and logistic regression models, averaged word vectors were used for GloVe and ELMo embeddings. Similar to bag-of-words (BOW) models, all of the word vectors in a given sentence were averaged element-wise to produce the final input vector. While the BOW-like representation does not explicitly model word order, in the case of ELMo, word order context may be encoded in the vector values. For BERT embeddings with the fully connected model, we chose to use the BERT sentence vector generated during the embedding stage. This was chosen over the average BOW method as the sentence vector does explicitly depend on word order, hopefully allowing it to contain more information than the average word vector.

For the convolutional neural network, the word vectors in the sequence were concatenated to form a single input tensor. Sequences shorter than 32 word / token vectors were padded with zero vectors, while sequences longer than length 32 were truncated, creating uniform inputs of size $32 \times d$, where d is the embedding size. Analysis of the training data showed that 97% of the training sequences were of length 32 of shorter, with 99%+ of training sequences under length 40. Based on this information, a size of 32 was chosen as a compromise between complexity and information retention.

6.3.1. Logistic Regression

A generic logistic regression model was used to obtain a baseline performance for each of the embeddings. Each model was trained for 3000 steps using a stochastic gradient descent optimizer with cross-entropy loss, a batch size of 512, and a learning rate of 0.001. The logistic regression model was trained on averaged GloVe and ELMo vectors, as well as BERT sentence vectors.

6.3.2. Fully Connected Neural Network

The fully connected model consists of 2 dense layers with batch normalization, followed by 1 dense layer with a softmax layer. The full architecture can be seen in Figure 2. Each model was trained for 3000 steps using the Adam optimizer with cross-entropy loss, a batch size of 512, and a learning rate of 0.001. The fully connected network was trained on averaged GloVe and ELMo vectors, as well as BERT sentence vectors.

6.3.3. Convolutional Neural Network

The 1-dimensional convolutional neural network consists of three convolutional layers followed by max pooling layers and a final linear layer. Convolutions are performed across the padded and truncated sentence sequences, with each dimension of the vector embeddings treated as a separate channel. The first convolutional layer has a filter size of 5, stride of 2, and 32 filters. The second and third convolutional layers each have a filter size of 3, with stride 1 and 64

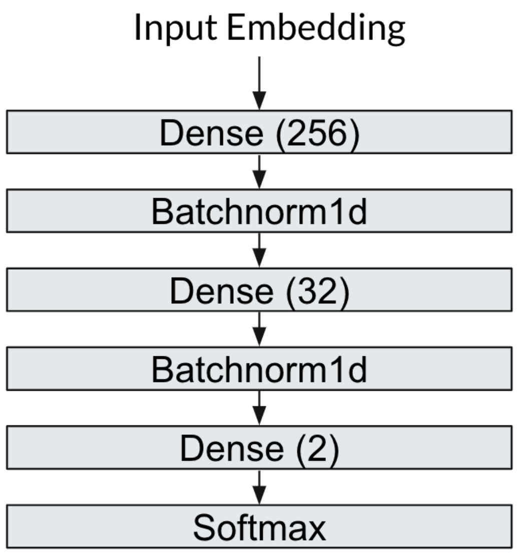

Figure 2: Network architecture for the fully connected neural network

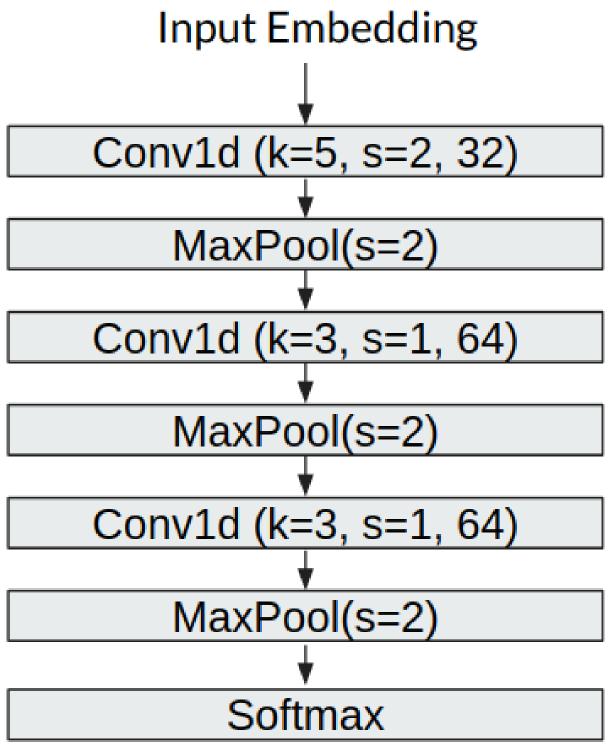

Figure 3: Network architecture for the convolutional neural network

filters. The full architecture can be seen in Figure 3. Each model was trained for 3000 steps using the Adam optimizer with cross-entropy loss, a batch size of 512, and a learning rate of 0.001. The convolutional neural network was trained on padded and truncated vector sequences for GloVe, ELMo, and BERT embeddings.

7. Dataset

In this project, we use the dataset provided by the Quora Insincere Questions Kaggle competition[3]. It consists of 1.31 million labeled training examples, and 375,806 unlabeled test examples. Each labeled example includes the raw, unpreprocessed text of a user-submitted Quora question, and a binary target representing the true label. In particular, a label of 1 (positive example) represents an ”insincere” question, while a label of 0 (negative example) represents a legitimate question. Examples of positive and negative training data can be seen in Table 1. The dataset also contains four pretrained embedding files, which are formatted as lookup tables that convert lowercased words to their corresponding vector representations. The four types of word vectors that are provided are vectors generated with word2vec, GloVe, Paragram, and FastText.

7.1. Pre-processing

Initial inspection of the dataset shows that the label classes are unbalanced: 94% of the questions are negative examples (legitimate questions), while only 6% of them are positive examples (”insincere” questions). As a result, we use weighted sampling during the training process, to make sure that each training batch contains an equal proportion of positive and negative samples.

For this project, we use solely the 1.31 million labeled training examples from the Kaggle dataset; this is because we have no way of evaluating our model performance on the unlabeled examples. During model training, we split this labeled dataset so that 70% is used for training, 10% is used for validation, and 20% is held out for testing / evaluating the trained models.

Depending on the word embedding type used, we preprocess the data differently, according to the previ-

Sincere	How did Quebec nationalists see their province as a nation in the 1960s? Why does velocity affect time? Does velocity affect space geometry?
Insincere	How do I marry an American woman for a Green Card? How much do they charge? Did Julius Caesar bring a tyrannosaurus rex on his campaigns to frighten the Celts into submission?

Table 1: Examples of sincere and insincere question from the training data.

ous Input Preprocessing section.

8. Experiment Results

The F1 score results for the embeddings and classifiers combinations are shown in Table 2 below. The highest overall performance was achieved with ELMo embeddings and the convolutional network. ELMo embeddings with both the fully connected and convolutional networks outperformed all other embedding and model combinations. ROC curves comparing embeddings can be seen in Figure 4, 5, and 6.

We see that in general, fully connected neural networks and convolutional networks result in a higher F1 score than logistic regression, with the CNN 1D giving better performance than the fully connected network. We also see that ELMo embeddings give better performance with FC and CNN models than BERT embeddings do, but result in poorer performance than BERT when used with the logistic regression classifier.

We can see this same trend from the receiver operating characteristic (ROC) curves. The better performing classifier / embedding combinations are closer to an ideal ROC curve, and have a higher area under the curve (AUC). We can see that the best performing classifier / embedding combination, ELMo + CNN1D, resulted in an AUC of 0.9760.

	Log. Reg.	FC NN	CNN1D
GloVe	0.000010	0.508216	0.592245
ELMo	0.092186	*0.609533*	***0.667888***
BERT(Token)	—	—	0.588146
BERT(Sentence)	*0.386271*	0.518742	—

Table 2: F1 Scores for logistic regression, fully connected neural network (FC) and convolutional neural network (CNN 1D) models on GloVe, ELMo, BERT embeddings.

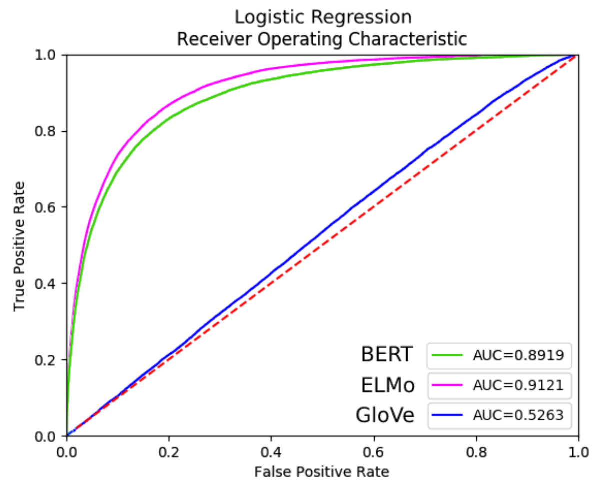

Figure 4: ROC curves for logistic regression with different embeddings

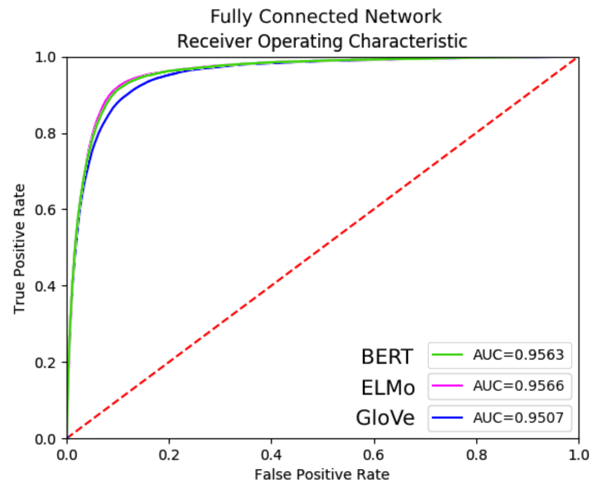

Figure 5: ROC curves for fully connected network with different embeddings

9. Discussion

As previously discussed, the state-of-the-art model achieved an F1 score of 0.71323 on GloVe and

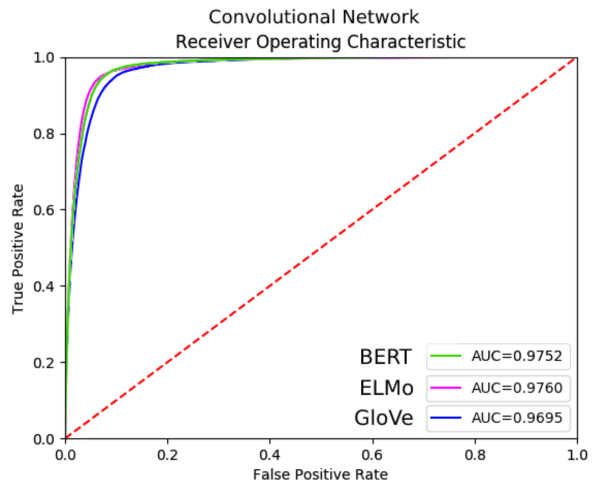

Figure 6: ROC curves for convolutional network with different embeddings

Paragram embeddings. Our best performing model, CNN1D + ELMo embeddings, only achieved a maximum F1 score of 0.66788. However, while the performance of the best model explored in this work did not exceed the performance of the previous state-of-the-art model on this dataset, our explorations did show the advantages of using context-dependent embeddings over previously used embeddings. In addition, the majority of other classifiers in the Kaggle competition achieved F1 scores between 0.65 and 0.7. So, using a single convolutional neural network classifier and the ELMo pretrained embeddings, we achieved similar performance to many existing models in the Kaggle competition using a drastically simplified architecture.

In this work, in addition to not using RNN based architectures, we did not use the statistical features or ensemble methods utilized by numerous Kaggle submissions, including the state-of-the-art. Given more time, in combination with more thorough hyperparameter tuning, these augmentation could possibly allow us to achieve performance near the state of the art with our model. Prior work[2] as well as discussion in the Kaggle competition results have shown that model ensemble results in significant improvements in F1 score, so to achieve the results we did shows the advantage of context-dependent embeddings over context-free ones.

One surprising result of our explorations was the relatively poor performance of the BERT embeddings when compared to ELMo. In many cases, BERT was only marginally better or worse than the context-free GloVe embeddings, and much worse than the ELMo embeddings. This lack of performance was surprising given results from past works showing BERT to have state-of-the-art performance on multiple natural language processing tasks. However, there are several plausible explanations for this discrepancy. First, due to time and resource constraints, the models explored in this paper were only trained for approximately 3000 steps, meaning that the models may not have had time to fully converge to their optimal parameter settings. Second, we used the BERT sentence embeddings for the logistic regression and fully connected embeddings, instead of averaging the sequence of token embeddings, like we did with the ELMo word embeddings. This could have resulted in performance differences. Additionally, the BERT transformer can be fine tuned to specific tasks, and most previous works utilize this fine tuning. Given the resource requirements, fine tuning the BERT transformer was not a viable option for this project, and this omission may have contributed to the relatively poor results for the embedding. Finally, we can see from the ROC curve that the AUC for the BERT model was greater than the of GloVe, and nearly the same as ELMo. This suggests that despite the lower F1 score, the BERT classifier might be performing well. Overall, the high relative performance of the BERT embeddings with the logistic regression model prove promising, and suggest that with further exploration BERT performance could be improved on other models.

We hypothesize that the CNN1D with the ELMo model resulted in the best performance for several reasons. We saw that ELMo achieved better performance than BERT; and one of the primary differences between ELMo and BERT is that ELMo operates on the raw unprocessed question text while BERT preprocesses it. This means that ELMo incorporates information about ”statistical features” like capitalization, special characters and emojis, which the state-of-the-art model demonstrated were important to classification. BERT on the other hand, removes all this information during preprocessing. This could be one of the reasons why the CNN1D achieved better performance on ELMo rather than BERT.

Our results also justified our use of F1 score as a quality metric when evaluating our models. All of the models trained in this work achieved greater than 90% accuracy, but given the composition of our dataset, achieving high accuracy alone is meaningless, as a trivial classifier always classifying questions as ”sincere” could achieve a 94% accuracy. Because F1 score takes into account both precision and recall, it provides a better assessment of a models ability to differentiate between positive and negative examples. especially in an unbalanced dataset such as the one used in this paper.

10. Conclusion

Overall, while the performance of simple models with context-dependent embeddings were not able to match the high performance of more complicated RNN based architectures, our results show the advantage of context-dependent embeddings in NLP classification tasks.

In many tasks the use of context-dependent embeddings may allow users to trade a small hit in performance for a large decrease in training time and required compute power. Even in performance critical tasks, the use of context-dependent embeddings in addition to RNN based architectures has the potential to achieve state-of-the-art performance in toxic question classification and other NLP classification tasks.

Appendix

Code for the models described in this paper can be found here.

References

[1] Spiros V. Georgakopoulos, Sotiris K. Tasoulis, Aristidis G. Vrahatis, and Vassilis P. Plagianakos. Convolutional neural networks for toxic comment classification. *CoRR*, abs/1802.09957, 2018.

[2] Betty van Aken, Julian Risch, Ralf Krestel, and Alexander Löser. Challenges for toxic comment classification: An in-depth error analysis. *CoRR*, abs/1809.07572, 2018.

[3] Kaggle. Quora insincere questions classification, 2018.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[6] Balázs Csanád Csáji. Approximation with artificial neural networks.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.