Offensive Tweet Classification via Various Deep Learning Techniques

Rowan Lavelle and **Radheshyam Verma**
Indiana University
Graduate Luddy School Of Informatics, Computing and Engineering
rowan.lavelle@gmail.com, radhe2205@gmail.com

Abstract

Classification of offensive text on social media is a current and difficult problem trying to be solved by the NLP community. In this paper we present experiments and comparisons for multiple deep learning architectures. We propose the use of a self attention layer within an LSTM network as a valid structure for time dependent classification. This method is tested across all tasks posed in (Zampieri et al., 2019) to show that the attention layer increases performance of the model. The semi-supervised tri-learning ensemble method is compared across many different sets of models to show its strengths when mass unlabeled data is available, and also when it may fail. Finally we attempt to show plausibility of using transfer learning by training base models on tasks not included in (Zampieri et al., 2019) and using the learned information to aid in classification across the posed tasks.

1 Introduction

As the world of social media continues to grow larger and larger, attracting a broader and more diverse user base every day, logistical issues come up rapidly. One of the larger problems social media faces right now is the spread of disinformation and highly offensive content.

In this paper we will be attempting to tackle this problem through various deep learning architectures. Some of the main goals are to be able to classify offensive tweets, and make a generalized model that works over a wide set of data. We will be attempting to answer three questions in this paper

(a) Can the use of a self attention layer on a bidirectional LSTM provide increased accuracy in classification.

(b) Can tri-learning be used effectively as an ensemble method, and generalize to work on new data.

(c) Can transfer learning be used to improve overall model performance.

Data Distribution							
	Task A		Task B		Task C		
	NOT	OFF	TIN	UNT	IND	GRP	OTH
	OLID						
Train	0.66	0.34	0.88	0.12	0.63	0.27	0.10
Test	0.72	0.28	0.89	0.11	0.47	0.37	0,16
	SOLID						
Test	0.72	0.28	-	-	-	-	-

Table 1: Distribution of OLID/SOLID train and test data

2 Data

2.1 The Dataset

We use the OLID dataset (Zampieri et al., 2019) and the SOLID dataset (Rosenthal et al., 2020).

The OLID dataset is split up into multiple different tasks

(A) Offensive Language Detection

(B) Categorization of Offensive Language

(C) Offensive Language Target Identification

For task A the labels are (NOT, OFF), for not offensive and offensive tweets. For task B the labels are (TIN, UNT) for targeted and untargeted offensive tweets, finally for task C the labels are (IND,GRP,OTH) for targeted at an individual, a group, or other[1].

The OLID data was collected by scraping certain types of hashtags that are known to have lots of offensive comments (political posts, searching by offensive key words etc). The class percentages for each tasks train and test sets can be seen in table 1.

The second dataset that we use is the SOLID dataset. The SOLID dataset contains the same hierarchical label set as OLID, however a semi supervised process was used to label the training

[1]a group in this case encompasses ethnicity, gender, sexual orientation, political beliefs, religious beliefs etc.

data which was collected randomly based on stop words (Rosenthal et al., 2020). To label the data multiple architectures such as BERT and FastText were tasked with giving predictions on unlabeled tweets, then the confidence scores were averaged to create the labels.

An issue with the SOLID dataset is that it contains highly offensive tweets, many of the tweets in the dataset no longer exist, either due to private accounts, user bans, or user removal of tweets.

One upside to the SOLID dataset is that there are 9 million tweet IDs available, and after scraping 250,000 of them $\approx 25,000$ were available for use at a perfect 50/50 split on classes for training.

2.2 Data Preprocessing

Due to the complicated nature of this problem, extensive preprocessing needs to be done. The goal is to generalize the tweets as much as possible. This gives us the best chance of having our model generalize well on outside data. For preprocessing we do similar steps to (Cambray and Podsadowski, 2019).

The preprocessing involves removing user mentions, URLs, stop words, non ascii characters and numbers. We also lowercase the data, handle apostrophes to split up words, reduce words such as "reallllly" to "really", spelling correction, lemmatization, and sentence padding.

Tests were performed such as keeping stop words, or keeping hashtags, or having all words lowercase. In the end it was easy to see that using all of the above steps helped the model generalize and test well on both OLID and SOLID data, for this reason these miscellaneous results have been omitted from the paper. A value of $p = 50$ is used in (Cambray and Podsadowski, 2019) however we found using a value of $p = 70$ produced better results.

For experiments involving attention layers and (Cambray and Podsadowski, 2019) implementation (Pennington et al., 2014), we found that glove has embeddings for special characters and punctuation's, so we do not remove special characters for those experiments and add a space around them to be used as a token. We found that this performs better. All stop words are also kept.

3 Related works

Due to the nature of this problem there is a plethora of existing work using deep learning as a classification system for offensive language. Since we are focused on using the OLID and SOLID datasets, there are a good amount of papers for the competition every year.

The paper that we will be comparing against is the same one as the above preprocessing methodology (Cambray and Podsadowski, 2019). We will implement the best performing model from the paper, which is bidrectional LSTM. The model architecture is shown in the figure 4.1.1.

The approach from (Frisiani et al., 2019) is similar to (Cambray and Podsadowski, 2019), they attempt to use a CNN as the output feature extractor taking in values from an LSTM the results are very similar even with a change in architecture.

A simpler approach is taken in (Sapora et al., 2019) which uses a logistic regressor with random draws for up sampling as its baseline and best performing model. They were able to achieve f1-scores of 0.85 to 0.9 on testing, however these results are suspicious considering they use test data in conjunction with training to create their one hot encodings.

Because of their success we attempted to use up sampling methods, both random draws and SMOTE. Both up sampling techniques generated worse results, and did not improve generalizability. We omit these results from the paper.

Finally (Uglow et al., 2019) uses transfer learning to retrain BERT on the OLID dataset and has good results across the board, however these results are comparable to the models in both (Frisiani et al., 2019) and (Cambray and Podsadowski, 2019). Related works show more complex models struggle with the tasks

4 Methods

4.1 Self-Attention and BiLSTM Model

We implement Bidirectional LSTM model from the Aleix et al. The architecture of the model is shown in the figure 1.

LSTM models may face difficulty in coming up with relations of each word with the other word. We believe if the model has access to sentiment around each word, it may help it predict the target label better. With this in mind, we use self-attention on the LSTM layer output of the tweet words and pass that through another LSTM layer in many-to-one configuration to predict the target label. We use multi-head attention(Vaswani et al., 2017), with 4 attention heads. The architecture of the model is shown in figure 2.

Model name	learning rate	batch size	num_layers	Seq Len	dropout	Gamma	hidden dim	Embedding dimension	Word | Char model
Aleix et al.	0.001	32	1	50	0	0.99	50	300	Word
Simple LSTM	0.001	32	1	70	0.5	0.99	128	300	Word
Deep LSTM	0.001	32	2	70	0.5	0.99	128	300	Word
Grid Searched LSTM	0.0001	32	2	70	0.5	0.95	32	300	Word
Chracter LSTM	0.001	32	2	-	0.5	0.95	128	50	Char

Table 2: Model parameters for tri-learning

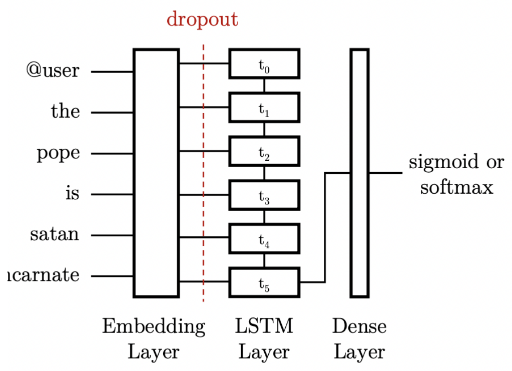

Figure 1: BiDirectional model architecture

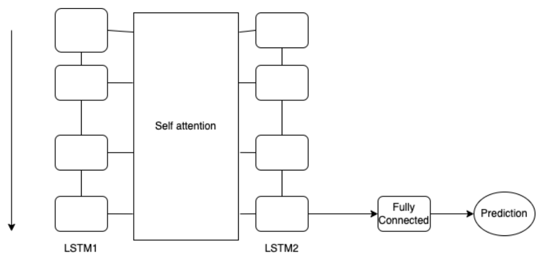

Figure 2: Self-Attention Model architecture

4.2 Tri-Learning

Tri-learning is a semi-supervised ensemble machine learning technique than can allow for the use of unlabeled data to be used as training data for a problem (Dong-DongChen and WeiGao, 2018). For our purposes we have two datasets, OLID and SOLID. The methodology here is to train and optimize three different machine learning models using the OLID training data and then perform a few experiments. For the purpose of these experiments, results were found for task A only.

(a) Use all three trained models to predict labels for N SOLID training data points. Using these predictions we take a majority vote and use this as the label and add the point to the training set. Using this new larger dataset, we retrain all three models and test on both OLID and SOLID test sets.

(b) Take a threshold of the confidence scores to assign a label for N SOLID training points. Add these to the training set, then retrain all three models on this larger training set and test on both OLID and SOLID test sets.

(c) Repeat the above experiments using different values of N to see what the affects of adding more data are.

The above experiments were run across a plethora of ensemble models, five are selected for this paper, and the best ensemble of the five was selected to perform experiment (c). Here we try to create a balance in models between simplicity, complexity, and diversity to show a range of effects caused by tri-learning.

For all experiments a validation set of 10% of the training data was used to perform early stopping. The early stopping method computes a simple moving average of the f1-score to cut training.

For all of the models in this section the hyper parameters can be found in table 2.

4.2.1 Reconstruction of Aleix et al. Baseline LSTM as an Ensemble

For our first proposed model for tri-learning, we reconstruct the baseline bidirectional LSTM model presented in (Cambray and Podsadowski, 2019) as an ensemble.

The goal behind this model is compare how the baseline LSTM model can perform when used as an ensemble and exposed to more data.

4.2.2 Simple LSTM as an Ensemble

Our second proposed model for tri-learning is an augmentation of the (Cambray and Podsadowski, 2019) baseline LSTM, this model is denoted as out simple-lstm.

The goal behind this model is to add slightly more complexity to (Cambray and Podsadowski, 2019) without becoming so complex that we become easily exposed to overfitting. The batch normalization layer and dropouts are used in an attempt to avoid the vanishing gradient issue.

4.2.3 Deep LSTM as an Ensemble

Our third proposed model for tri-learning is a more complex deep learning LSTM model. We denote this model as deep-lstm. Different than the other models, this model uses two dense layers on top of the recurrent layers.

The goal behind this model is to further add complexity to our ensemble to see if we can push results to become better. In (Cambray and Podsadowski, 2019; Frisiani et al., 2019; Uglow et al., 2019) convolution layers are stacked on top of the recurrent layer, but each of these papers seemed to note a decrease in accuracy when doing this due to overfitting. The intuition behind the use of the second dense layer is to add more complexity to the model, But also at the same time attempt to avoid the overfitting problem shown in the mentioned papers.

4.2.4 Grid Searched LSTM as an Ensemble

Our fifth proposed model for tri-learning utilizes a grid search to find the best structure and hyper parameters. The grid search maximizes f1-scores of models over 768 different parameter combinations.

Namely we search over different number of hidden LSTM layers, different recurrent layer sizes, different learning rates, different batch sizes, and different GloVe embedding sizes.

For this specific model when the grid search produced a lower learning rate, we experimented with the same learning rate for the previously proposed models, however this lowered results[3].

4.2.5 Diverse Selection of Models as an Ensemble

For our final proposed tri-learning model we experiment with diversity in the ensemble to see what the effects of tri-learning are. This ensemble is made up of two LSTMs and a Logistic Regression model. The first LSTM model is the same as the one listed above, which was produced from the grid search, and the second LSTM model is a character level model.

The hyper parameters listed for this model were found through a hand tuning process.

For this model we pad all strings to be the same length as the maximum string in the training set with a pad token.

The goal behind this model is to take a different approach than the rest of the models proposed. Character level LSTMs can be very powerful, they learn different patterns and correlations than word level embeddings and word level models.

[3]experiments for lowering γ were done for each proposed model, the value of γ that resulted in the best model is what is shown

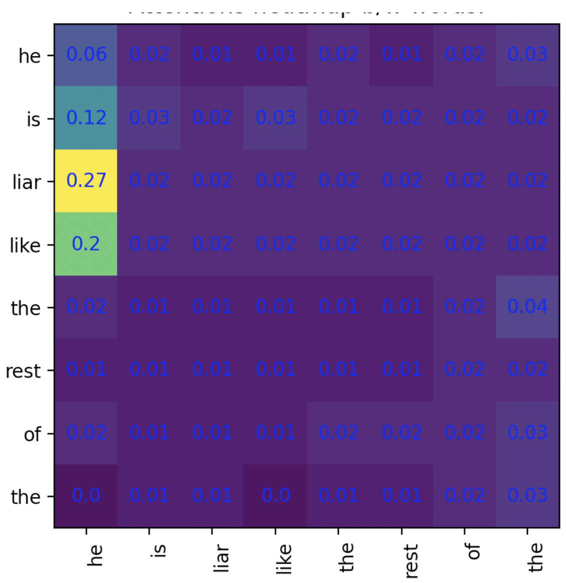

Figure 3: Attention Heatmap

Finally the inspiration for using Logistic Regression is that its simple, and it was seen to produce good results in (Sapora et al., 2019).

The Logistic Regression model is composed of a single dense layer using sigmoid activation. This is paired with binary cross-entropy loss, and the model uses stochastic gradient descent with momentum for training. The model uses a batch size of 32, a learning rate $\eta = 0.01$ and $\gamma = 0.95$ exponential decay for η. Hyper parameters for this model were found through a hand tuning process.

For this model we experimented with multiple different feature extraction methods to generate input. Trainable embeddings, TF-IDF vectors, and count vectorization were compared. After experimentation, it was found that using count vectorization performed best, this result was also found in (Sapora et al., 2019).

Count vectorization uses word ngrams of sizes (1,3), and we keep the entire feature space.

4.3 Transfer Learning

Transfer learning can be described as the transfer of knowledge from one task or domain over to another task or domain where there may be a commonly shared feature space (Pan and Yang, 2010). In the case of this paper we are looking to experiment on if its possible to build models on a tasks not described in the OLID paper, then use these models as a base to retrain new stacked layers to classify across OLID tasks A, B and C.

4.3.1 Language Model Based Transfer Learning

Our first proposed method to experiment with transfer learning is through language modeling. The goal here is to train a deep LSTM through teacher forcing to predict the next word given an input sequence of words.

For training we sequentially turn the entire dataset (after cleaning) into a list of words to create our corpus. After we randomly choose a window size between 5 and 10 to be the input sequence. we then attempt to predict the next word from the sequence. This process is repeated for each epoch.

The structure and hyper parameters for this language model are as follows.

- 2 recurrent layers
- 256 recurrent hidden units
- 0.001 learning rate η
- $\gamma = 0.95$ exponential decay for η
- 300 dimensional trainable embeddings

The recurrent layer for this model is not bidirectional, output from the recurrent layer is passed to a dense layer to generate the output.

Once this model is trained we save the learned embeddings and recurrent layer weights, these are the knowledge that we transfer to the new model. The intuition behind this is that the embeddings and recurrent layer will learn to model the language of the data, and will then provide useful information as a starting point for classification.

The transfer language model shares the same embeddings and recurrent layer weights as the language model. We then stack on our new classification layers, in essence our transfer models inputs are the out puts of the language model.

The transfer model classification layers have the following structure and hyper parameters (not including the embeddings and recurrent layers from the language model).

- 1 recurrent layer
- 32 recurrent hidden units
- 0.01 learning rate η
- $\gamma = 0.95$ exponential decay for η
- 32 batch size
- $p = 70$ sentence length for padding

The recurrent layer is bidirectional, output is fed to a dense layer. Depending on the task we use either a sigmoid activation, or the identity function.

When training the transfer model we load in the language model weights and set their learning rate to be very small ($\eta_{lm} = 0.00001$), and then use the given value of η to train the top layer.

Model	Loss Type	Task A		Task B		Task C	
		F1	Acc	F1	Acc	F1	Acc
Self-Attention	simple loss	**0.776**	**0.825**	0.594	0.804	0.481	0.571
	weighted loss	0.747	0.800	0.592	0.774	**0.494**	0.586
Paper	Weighted loss	0.738	0.780	**0.61**	0.57	0.50	**0.69**
BiLSTM paper impl	Weighted loss	0.747	0.783	0.602	**0.808**	0.478	0.578
	simple loss	0.755	0.808	0.578	0.790	0.485	0.584

Table 3: Aleix et. Al results

4.3.2 Multi-Class Classification Based Transfer Learning

The second proposed method to experiment with transfer learning is through aggregated classification. The goal here is to train a deep LSTM on classifying multiple classes across tasks at the same time.

Each point in the OLID dataset has three labels, one for each task. Using this we can consider each unique tuple of classes[4] as a class itself. This results in a 5 class classification problem.

The structure and hyper parameters for this multi-class model are as follows.

- 2 recurrent layers (0.5 dropout)
- 256 recurrent hidden units
- 0.0001 learning rate η
- $\gamma = 0.95$ exponential decay for η
- 32 batch size
- 300 dimensional GloVe embeddings
- $p = 70$ sentence length for padding

The recurrent layers are bidirectional, output is fed to a batch normalized dense layer. Since we use cross-entropy loss so there is no need for an activation function.

For training purposes we take the OLID training set and use 10% as a validation set, and then 20% of training set as the test set. The hyper parameters of this model were tuned by hand.

Once this model is trained we save the recurrent layers to be used in the transfer model. The intuition behind this approach is that if we can build a model that can somewhat differentiate between the aggregated classes, it will have useful knowledge about all tasks allowing us to then extrapolate off that to classify for only one task.

The multi-calss transfer model uses the pre-trained GloVe embeddings and recurrent layer from the multi-class model and stacked additional layers on top. To remain consistent we use the same stack

[4]as an example (OFF,TIN,GRP) could be a tuple of classifications for one tweet

Model	**Task A**		**Task B**		**Task C**	
	F1	Acc	F1	Acc	F1	Acc
Attention	**0.778**	**0.8281**	**0.617**	**0.847**	**0.497**	**0.595**
Two LSTM	0.758	0.809	0.542	0.750	0.479	0.580
Attention with concatenation	0.755	0.807	0.581	0.789	0.477	0.569

Table 4: Attention layer experiment results

structure, hyper parameters, and training procedure as the transfer language model.

5 Results

For each of our methods and results we report overall model accuracy and macro averaged f1-score.

5.1 Self-Attention & BiLSTM Results

In our experiments, the results vary between different runs of the model. So we run our model five times and take average of all metrics observed in different runs.

We observe that self attention model performs slightly better than just using BiLSTM. Results using self-attention layer are listed in the table 3. We see that the attention model also uses two LSTM layers, while the BiLSTM uses only one LSTM layer. To investigate whether the attention layer adds the goodness or the BiLSTM layers before and after the attention layer, is solely contributing to the results, we conduct further experiments. In experiment 1, we use the attention layer between two LSTM layers to make a prediction. In experiment 2, we remove the attention layer and just use 2 BiLSTM layers to make a prediction, while everything else remains the same. In 3rd experiment, we concatenate the result of first LSTM layer and attention layer and pass it to second LSTM layer. The results of this experiment are listed in the table 4 We observe that attention model performs better.

We investigate the attention heatmap of words, to see if the attention model understands the relations between the words. Figure 3 shows the attention heatmap grid of the words. We observe that the attention model, largely can make out the relations. However, twitter grammar and slightly smaller dataset size, makes it harder for attention model to model word relations accurately. In the figure 3, we see that the attention on the words representing individuals, seems to be working fine. The sum of the attention in the image does not add to one because the attention on the padding tokens is not shown. It was also observed that attention on some important words was small(0.03) but it is still high compared to other words. However this small attention might not be as useful, because in attention model we combine attention from all the words, so the there is very little contribution of the important word.

5.1.1 Experiments with embeddings, weighted loss

In these experiment, we implement the (Cambray and Podsadowski, 2019) paper. We tried three combinations of embeddings. In first experiment we used custom trained embeddings for the words. In second experiment we used pre trained glove embeddings. In the third experiment we used a mix of both, where we created trainable embeddings of the words which occur more than 1 times in training dataset and are not present in glove embeddings, in combination with glove embeddings. Our intuition was that it may help model learn the twitter language better.

We found that using just glove embeddings performs better than all other combinations. It performs even higher than using hybrid (trainable + pre trained) embeddings. We believe this may be due two reasons. First, the trainable embeddings cause more confusion as these embeddings do not contain multi-faceted information and model has no way to distinguish whether the embeddings that are being given to it are trainable. Second, the trainable words do not occur that often in the tweets. Which means they are very infrequently back-propagated. As we train the model for 60 epochs, it does not give the trainable embeddings enough time to be trained. So it becomes a race between the model over-fitting to the train data vs trainable embeddings reaching a better representation.

In our next experiment, we use weighted loss, to tackle the class imbalance. For this, we used the methodology of (Cambray and Podsadowski, 2019). We found that weighted loss does not produce significant change. We observe that weighted loss works better for task C while works worse for task A and B. This could be due to more classes and greater imbalance in the dataset for task C.

Results of our implementation of the (Cambray and Podsadowski, 2019) paper with and without weighted loss along with multi-head attention layer are listed in table 3.

	Aleix et al. Ensemble		SimpleEnsemble		DeepEnsemble		GridEnsemble		DiverseEnsemble	
	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy
Experiment	OLID test set									
(a)	0.7234	0.8086	0.7407	0.8016	0.7671	0.8216	**0.7673**	**0.8274**	0.7180	0.8086
(b)	0.7570	0.8309	0.7515	0.8134	0.7538	0.8133	**0.7619**	**0.8298**	0.7134	0.8157
(c)	0.7639	0.8063	0.7765	0.8169	0.7766	0.8192	**0.7913**	**0.8321**	0.7722	0.8286
Experiment	SOLID test set									
(a)	0.8671	0.8925	0.8668	0.8889	0.8776	0.8969	**0.8894**	**0.9080**	0.8699	0.8977
(b)	0.8717	0.8964	0.8803	0.9010	**0.8908**	**0.9080**	0.8854	0.9057	0.8529	0.8892
(c)	0.8997	0.9134	0.8962	0.9103	0.8963	0.9101	0.8985	0.9123	**0.8997**	**0.9144**

Table 5: Tri-learning ensemble results

	Experiment (b)				Experiment (c)			
	OLID test set		SOLID test set		OLID test set		SOLID test set	
	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy
N=1,000	0.7227	0.8122	0.8629	0.8912	0.7442	0.8204	0.8741	0.8979
N=5,000	0.7205	0.8122	0.8597	0.8892	0.7806	0.8321	0.8936	0.9090
N=10,000	0.7540	0.8298	0.8781	0.9011	0.7826	0.8286	0.8948	0.9095
N=15,000	0.7619	0.8298	0.8854	0.9057	0.7913	0.8321	0.8985	0.9123
N=30,000	0.7275	0.8169	0.8662	0.8933	0.7832	0.8286	0.9038	0.9167

Table 6: Varying of N SOLID training points for tri-learning

5.2 Tri-Learning Results

The f1-scores and accuracy's for tri-learning over each experiment can be seen in table 5.[5] A more in depth look at the individual model scores over each experiment during tri-learning for the diverse ensemble can be seen in table 6. Finally the change in performance over varying the number of SOLID samples used in tr-learning can be seen in table 5. This experiment was done using the grid searched LSTM ensemble. For table 6 experiment (a) is omitted because the change in solid training size has no effect.

The simple LSTM ensemble, grid searched ensemble, and deep LSTM ensemble use bidirectional recurrent layers to feed output to a dense layer. The dense layer uses sigmoid activation, batch normalization and a dropout of 0.4. Training is performed through bootstrapping[2] for these models. The deep LSTM however feeds to a second dense layer with the same stack as listed above. The (Cambray and Podsadowski, 2019) ensemble uses the above structure but without batch normalization.

[5]note that experiment (a,b,c) is referencing the experiments denoted in the tri-learning methods section.

[2]a seed is set independently for each model so that the bootstrapping returns the same training data between experiments to remain consistent.

5.2.1 Model Comparison

As seen table 5, our (Cambray and Podsadowski, 2019) ensemble performs well, and performs better than the reported results in (Cambray and Podsadowski, 2019) when using predicted SOLID labels, and thresholded labels in training. In experiment (b) we get an f1-score of 0.757 and an accuracy of 0.8309, and in experiment (c) we get an f1-score of 0.7639 and an accuracy of 0.8063. (Cambray and Podsadowski, 2019) reports an f1-score of 0.7382 and an accuracy of 0.7801 for task (a).

When testing on OLID comparing all the ensembles, the best performing ensemble is the grid searched LSTM ensemble across each experiment. When testing on SOLID there is a larger variety of results. Before any SOLID data is seen, the grid searched LSTM ensemble performs the best, when using predicted labels the deep LSTM ensemble performs the best, and when using the thresholded labels the diverse ensemble and (Cambray and Podsadowski, 2019). ensemble perform almost identically, but the diverse ensemble has a marginally better accuracy.

There are a few interesting trends that can be seen from this table. When testing on OLID, looking at the (Cambray and Podsadowski, 2019). Ensemble and the simple LSTM ensemble, we see increases in f1-score and accuracy when using predicted SOLID labels. These two ensembles are our less complex models. In comparison, our more

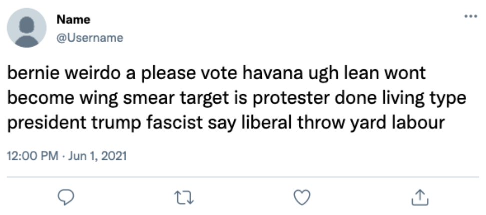
Name
@Username
bernie weirdo a please vote havana ugh lean wont become wing smear target is protester done living type president trump fascist say liberal throw yard labour
12:00 PM · Jun 1, 2021

Figure 4: Fake tweet generated by language model

Diverse Ensemble				
Experiment (a)				
	OLID test		SOLID test	
	F1-Score	Accuracy	F1-Score	Accuracy
GridLSTM	**0.7512**	**0.8122**	**0.8845**	**0.9036**
CharLSTM	0.6347	0.7124	0.7768	0.8217
Logit Reg	0.6854	0.7981	0.8556	0.8905
Ensemble	0.7180	0.8086	0.8699	0.8977
Experiment (b)				
	OLID test		SOLID test	
	F1-Score	Accuracy	F1-Score	Accuracy
GridLSTM	**0.7372**	**0.8263**	**0.8597**	**0.8920**
CharLSTM	0.6642	0.7817	0.851	0.8874
Logit Reg	0.6991	0.8040	0.8453	0.8848
Ensemble	0.7134	0.8157	0.8529	0.8892
Experiment (c)				
	OLID test		SOLID test	
	F1-Score	Accuracy	F1-Score	Accuracy
GridLSTM	**0.7755**	0.8192	0.8982	0.9121
CharLSTM	0.7449	0.8052	0.8871	0.9036
Logit Reg	0.7490	0.8181	0.8954	0.9119
Ensemble	0.7722	**0.8286**	**0.8997**	**0.9144**

Table 7: Full tri-learning results for Diverse Ensemble

complex model the deep LSTM shows a decrease in f1-score and accuracy.

The most likely reason behind this that the deep LSTM ensemble is either overfitting, or the bootstrapping from the data is learning very similar models.

We can also see that when using SOLID thresholded labels, all models show an improvement in performance, this is most likely due to the additional data.

5.2.2 Generalizability

Table 5 shows that our grid searched LSTM is generalizing well, before seeing an SOLID data we get very good scores on the SOLID test set considering the imbalance in the classes shown in table 0. We also know this is generalizing well because (Rosenthal et al., 2020) collected their data differently than (**?**).

Table 5 also shows that our diverse ensemble, although it performs non comparably on the OLID test set, performs very well when given thresholded SOLID labels as part of its training.

Table 5 shows when testing on OLID, the character LSTM and logistic regression get small bumps in performance using predicted SOLID labels in training, but when thresholded SOLID labels are used both get substantial bumps in performance.

A different trend happens on the other side when testing on SOLID. When using predicted SOLID labels in training, the character LSTM gets a large performance bump, but the grid searched LSTM and logistic regressor take performance hits. When using thresholded labels each of the models perform similarly. This is very interesting, and most likely the reason why this ensemble out predicts the individual models, because of diversity there will be more disagreements within the majority vote for predicting.

As a final note for table 6 it should be observed that although the logistic regressor does not perform well at all on the OLID data when only trained on OLID, it performs very well on the SOLID data. This is evidence that the logistic regressor does a very good job of generalizing. This same trend occurs in the character LSTM.

5.2.3 Changes in training sizes

In table 6 we can see for experiment (c) when testing on SOLID, the more data we have the better we do, this seems to be increasing at a logarithmic rate given the slowing in f1-score increase as N increases. For testing on OLID a similar trend can be observed up until $N = 30,000$, the cause for this is uncertain, at this level there would be more than two times the number of SOLID training points than OLID training points. So although the quality of the SOLID thresholded data has been proven in tables 1 and 2, having too much of it may be diluting the niches of the OLID dataset.

On the other side of the table we see a somewhat similar trend, however there is more variance. The decrease in accuracy from 1,000 samples to 5,000s samples is most likely due to the random seed. The rest of the table however follows a similar explainable trend, at $N = 15,000$ the two datasets are relatively balanced, so this produces the best results for both testing on OLID and SOLID. Upping the number here again causes a decrease in performance in OLID testing, but also in SOLID testing this time. This is most likely because the quality of the predictions may not be good in this case, and

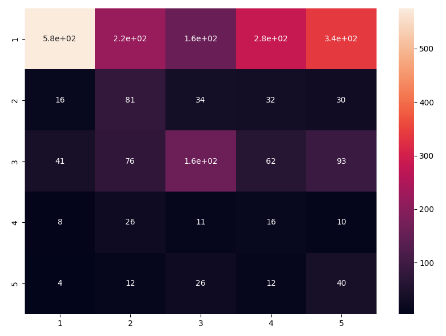

Figure 5: Confusion matrix for base multi-class model

that quality in turn makes the individual models learn many incorrect classes as correct, shifting their classification boundaries.

5.3 Transfer learning Results

A sample output from the language model is shown in figure 4 and the f1-scores and accuracy's of both the multi-class transfer model and transfer language model across all tasks are shown in table 8.

After much tweaking with the multi-class model which is used as the base to the multi-class transfer model, we were only able to achieve an f1-score of 0.2747 and an accuracy of 0.3686. This multi-class is very difficult in nature since we are attempting to separate the OLID dataset into its individual parts, if this model had a very high accuracy we would be able to predict if a tweet was offensive, targeted, and at a group all with one set of weights.

On the other hand the base language model seems to be doing a very good job of modeling the language in the dataset, a fake tweet can be seen in figure 4. It should be noted that the model was trained after all preprocessing had happened, so it is not capable of generating things like stop words and punctuation which would make the verbiage flow a lot better. This sample does however give insight that a lot of the tweets are highly political.

When comparing the two transfer models, it is very obvious that the multi-class transfer model does not work very well. Accuracy's for tasks A and B have been omitted for this model since the model was just predicting the majority class and not learning. For task C however the model performed better than the transfer language model.

To better understand why this is happening we can look at the confusion matrix for the base multi-

	Task A		Task B		Task C	
	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy
Transfer Language Model	**0.5591**	**0.6572**	**0.5779**	**0.86075**	**0.3490**	**0.3571**
Multi-class Transfer Model	0.4192	-	0.46979	-	0.4816	0.5095

Table 8: Transfer learning results

class model. This is displayed in figure 5. Note that the set of classes is mapped as follows, 1, (NOT, NONE, NONE), 2, (OFF, TIN, GRP), 3, (OFF, TIN, IND), 4 (OFF, TIN, OTH), 5 (OFF, UNT, NONE).

Here we can see that the model does okay when the tweet is not offensive, meaning it is null in tasks B and C. After it does okay predicting classes 2 and 3. During learning it may not be able to differentiate between offensive and targeted, but does learn some differentiation between group and individual. For this reason stacking more layers on top, to specifically learn among *group*, *individual* and *other* classes, there would be useful knowledge for separating between groups and individuals. Thus the model can have okay results when the transfer model is classifying task C.

This also explains why it only predicts the majority class for task A and B, for task B there is only one class in the base model for UNT, and for task A, NOT is where a majority of the classes lie, and will be in a separate space than the other 4 classes.

As for transfer language model, the results are not amazing but as a proof on concept the model performs alright, there is more work to be done in this regard.[6]

6 Conclusion

Offensive language classification is a complex, interesting, and difficult task to solve. Our approach tackles attempts at offensive language classification using a multitude of deep learning techniques and architectures. We propose the use of a self attention layer within an LSTM for the purpose of classification. We propose the use of the semi-supervised method tri-learning to use mass unlabeled data to help generalize and improve model performance individually, and as an ensemble. We also proposed the use of transfer learning to transfer knowledge from one task to another to increase model performance.

We found that self attention helps improve the results in language related tasks. We believe it can

[6]When cleaning out the code base and making sure testing and training worked, the language model got reset and now has similar results to the multi-class model. This is a mystery, retraining has not helped.

work even better on better grammatically structured documents and larger datasets.

We also found that a mix of trainable and pre-trained embeddings does not work well, results in more confusion and we also observed no significant difference using weighted loss in classification.

Our best tri-learning ensemble consisted of the LSTM produced through grid searching maximizing f1-scores and we show that such a model can generalize very well. Further we show that in tri-learning the use of too much unlabeled data can begin to have negative effects.

Within transfer learning we show that training a language model on the dataset and then stacking on additional layers can potentially work given a different structure or with finer hyper parameter tuning. We show that the multi-class model gives very interesting insights as to where in the high dimensional embedding states each class clusters together, and also why tasks A,B and C progress in difficulty due to nuance and data size.

For tri-learning it could be very interesting to see how the use of AdaBoosting could effect the model, this may help out greatly in tasks B and C for under represented classes since more weight will be applied to these misclassified points.

In future, we would want to explore better shared learning methods. As we believe a shared model that understands multifaceted language tasks, can perform better on all of the individual tasks. It may prevent the overfit to a single task, while understanding the language semantics better at the same time.

References

Aleix Cambray and Norbert Podsadowski. 2019. Bidirectional recurrent models for offensive tweet classification.

WeiWang Dong-DongChen and Zhi-HuaZhou WeiGao. 2018. Tri-net for semi-supervised deep learning. In *Proceedings of twenty-seventh international joint conference on artificial intelligence*, pages 2014–2020.

Nicolò Frisiani, Alexis Laignelet, and Batuhan Güler. 2019. Combination of multiple deep learning architectures for offensive language detection in tweets.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

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

Sara Rosenthal, Pepa Atanasova, Georgi Karadzhov, Marcos Zampieri, and Preslav Nakov. 2020. A large-scale semi-supervised dataset for offensive language identification. *arXiv preprint arXiv:2004.14454*.

Silvia Sapora, Bogdan Lazarescu, and Christo Lolov. 2019. Absit invidia verbo: Comparing deep learning methods for offensive language.

Harrison Uglow, Martin Zlocha, and Szymon Zmyślony. 2019. An exploration of state-of-the-art methods for offensive language detection.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*.