Irony Detection in English Tweets

Kyle Lavorato
Queen's University
Kingston, Canada
kyle.lavorato@queensu.ca

Monica Rao
Queen's University
Kingston, Canada
monica.rao@queensu.ca

Abstract

In this paper we present a deep-learning system constructed to solve the SemEval-2018 Task 3 "Irony Detection in English Tweets". We solve both subtasks of the problem, involving detecting if irony is present in a tweet and classifying that irony by type. We use Keras to create TensorFlow Dense LSTM networks with deep learning to accomplish this goal. The input to our system is cleaned and normalized with a custom slang preprocessing system. Both our text processing pipeline and model are available to the research community[1]. Also, we present an adaption of the BERT transformer architecture to this task for results comparison. Our results show an F1 score of 0.6500 and 0.5076 for Subtask A and B which would rank us 1st and 3rd in accordance with the official rankings of the competition.

1 Introduction

The analysis of sentiment and emotion in user generated content is a difficult task for machines in natural language processing. A special subset of this task is attempting to determine if the sentiment in the message indicates the presence of irony. With the rise of important figures such as politicians conversing through text on platforms like Twitter, it is becoming increasingly important to determine if they are making literal or ironic statements. This can also be extended to chatbots that converse with humans in support roles. They must be able to determine if a statement is ironic or sarcastic otherwise they will respond with incorrect sentiment, angering the user they are conversing with.

The problem of irony or sarcasm classification much like sentiment analysis is traditionally a binary problem. The determination of if a statement is ironic or literal is enough information to react accordingly. In SemEval2018 Task 3 (Van Hee et al., 2018) the problem is advanced through subtasks. Subtask A is the traditional binary classification of irony. Subtask B furthers the challenge by introducing three different classes of irony in addition to the option of literal classification. An ironic message can now either be ironic by polarity clash, situationally ironic or any other verbal irony class.

We begin by analyzing four of the unique possible solutions to this problem as presented by competitors in Section 2. We then discuss the dataset for our solution and our custom text processing pipeline to create our model's input in Section 3. Our approach to the problem is then explained in Section 4. Finally our experiment setup and the results of that testing are shown in Sections 5 and 6 respectively.

2 Related Works

The task of sentiment analysis is a classical natural language processing problem. There have been countless diverse strategies from different members of the research community, all centered around solving this task. Recently in this area, both natural language processing on tweets and attempting to determine ironic sentiment have become more popular goals.

With the conclusion of SemEval2018 in Janurary 2018, there were over 100 varying submissions in the attempts to identify and classify irony. Solutions differed in results, with the average overall F1 score ranking at approximately 0.592 for Subtask A and 0.375 for Subtask B (Eslefeve, 2018).

To contrast with our proposed solution, we have selected a subset of previous competitors to analyze their methods and strategies. The deep learn-

[1] https://github.com/KyleLavorato/Irony-Detection

ing models of unsupervised learning (Nozza et al., 2016) and Bi-LSTM (Marrese-Taylor et al., 2018) provide valuable insight on how to best structure the neural network for optimal learning. We also evaluate the feature based models of pun contrast (Mikhalkova et al., 2018) and affective content (Pamungkas and Patti, 2018). These solutions highlight the important features of the input text that can aid in determining ironic intent.

2.1 Unsupervised Irony Detection

The standard approach to irony detection uses popular techniques of classic supervised models. These models will normally distinguish and classify a sentence input as irony or sarcasm, where the input has been encoded by a pre-trained word embedding. Unsupervised models are a new and different variant of the traditional supervised model. They are a more complex deep model, using more layers of neural network, allowing the model to learn from an unlabeled dataset.

This system by Nozza et al. (2016) applies unsupervised irony detection. They accomplished this task by taking into consideration the topic-dependency of words by using a set of probabilistic topic models. These types of models then use statistical algorithms for discovering structures of the text body, and are useful for organizing the insights. This allows the authors to understand larger datasets and text bodies in their network. The system uses two types of these models, one being the TSM (Topic Sentiment Mixture) (Mei et al., 2007), which uses a mixture of topic sentiment predictions. The second model is JST (Joint Sentiment Topic) (Lin and He, 2009), which deals with language distributions.

In terms of the word embedding, the authors trained various neural networks, using the Skip-gram and CBOW models for irony detection. The training goal of the network in the CBOW model was to be able to combine the likeness of the surrounding words and be able to predict the words in the middle. Alternatively, the Skip-gram model's goal was to learn word vector likeness that would be able to predict the context in that same sentence.

The overall system was able to achieve results that out performed similar traditional supervised models. They demonstrate an accuracy increase of 13% F1 Score compared to the supervised approaches. This results in an overall F1 score average of 84.9% across all categories.

The approach of using an unsupervised model for irony detection appears to demonstrate better results and outperforms most supervised models. This also includes the benefit of allowing for a much larger dataset. Since the input is unlabeled data, a scraped dataset does not need to be manually annotated before it can be used as an input. The variety of complex neural networks that the unsupervised models use also appears to aid in improving the overall accuracy within the validation and test set scores.

While the unsupervised model appears to demonstrate good results it does not come without cost. The internal neural network models require a large quantity of time to implement, connect, configure and train. In addition, they require larger datasets as they must learn patterns on their own, without annotations. Therefore with such large scale data and time requirements, they may not be a useful solution for our SemEval2018 task.

2.2 Irony Detection Using Multilayered Bi-Directional LSTM

The works by Marrese-Taylor et al. (2018) in SemEval2018, approach irony detection through the use of a multilayered bi-directional LSTM model. This model is implemented without the use any external systems or libraries aside from the basic LSTM unit.

The model uses pre-trained word embeddings to aid in the multi-classification of irony into the three types. They implemented the model in python using pytorch to preprocess and resolve inconsistent features in the tweets. This eliminated issues such as varying lengths and sizes of inputs before they were encoded into the word embeddings. These processed word embeddings then are used for standard supervised training of the bi-directional model.

The authors experimented with different LSTM hyper-parameters and layer sizes, also adding a dropout layer to prevent over-fitting. They also perform a feature ablation to demonstrate that the pre-trained word embeddings did present a small positive impact on the overall F1 score. They were able to increase from an F1 score of 68.2% to 68.48% considering the sentence level features on the validation set. The system itself achieved an F1 score of 72.6% on the validation and 29.1% on the test set. Though the validation set out-

performed the SemEval baseline, the actual test scores indicate that the architecture of the model is unable to generalize under the current configuration. Therefore more work and research must be completed in order for this model to achieve validation-level results from the bi-directional architecture and additional embedded input information.

Overall the approach taken is a common solution in using pre-trained word embeddings with a bidirectional LSTM model for sentence analysis. It falls short in its performance, likely due to their embedding layer (embedding matrix) of the model of not assigning correct scaled values to compensate for their large dataset and for the manual labelling. From this approach we could adapt our system to a bi-directional approach. The bi-directional nature can aid in learning irony patterns, as some irony artifacts such as polarity contrast may be more articulate in the reverse direction. With our current feed forward network proposal this is information that will not be detected by the model. Although we must ensure that we do not repeat their mistakes in order for our model to generalize.

2.3 Detecting Irony Through Humour

A more unique approach to this irony detection comes from Mikhalkova et al. (2018) in their attempts in finding a universal algorithm to discover irony. Instead of a traditional deep learning method, they approach irony from a linguistic standpoint. They pick up on an inherit lexical ambiguity similarity between irony and the concept of puns. Consider the two sentences in Table 1.

Pun	I could tell you a chemistry joke, but I know I wouldn't get a **reaction**
Irony	Christmas shopping on 2 hours sleep is going to be **fun**

Table 1: Relationship in puns and irony

By standard human reasoning, we can easily determine these sentences to be either a pun or ironic. Although if you can examine them in more detail, they have a closeness to each other. The pun uses *reaction* to mean "chemical reaction" in a chemistry context, while meaning "emotional response" in a joke context. The ironic statement likewise uses the term *fun* which means "joy or amusement" in the statement context. Although it actually intended as the opposite meaning of "problem or trouble" in this context. Therefore both share the relationship of a statement with a coexistence of two topics, which a computer is able to detect.

The PunFields model (Mikhalkova and Karyakin, 2017) is used to perform the identification of the pun structures. Once a statement is identified by PunFields as a candidate, they disambiguate the lexicon to determine if it contains an opposition of meanings to indicate irony. Tweets are preprocessed, converted to vectors with *Tweet to Vector* and then fed into the system. They apply both Scikit learn with SVM and a deep learning network with Keras and TensorFlow for the identification and classification tasks.

The system was able to achieve F1 scores of 0.596 and 0.560 for the two models respectively. The authors state that the repurposed algorithm approach, while better than baseline systems, still has room for improvement. They state that system has difficulty with any puns where there are not two evidently different topics, ironic or not. Often the two groups are overshadowed by other noisy topics or the pun is heterographic, where the pun word is implied, causing a disbalance between the groups. These conditions in the base pun detection lead to the structure being missed and therefore cannot be classified as any irony type. Regardless, the authors have plans to fix this inadequacy by replacing the core of the system with more elaborate word2vec representations to match the missed cases. Their overall solution is totally unique in comparison with our proposed model. Nevertheless it provides useful insight on potential other ways to solve a difficult task.

2.4 Exploiting Structural and Affective Features for Irony Detection

Another feature based approach to irony detection is that of Pamungkas and Patti (2018). Unlike the previous solution, they focus on extracting sets of features and artifacts from the tweets to aid in classification. There is a focus on the structural and stylistic features, with additional capturing of affective resources present in tweets.

Structural and stylistic features in tweets are those that consist of lexical and syntactic features that classify aspects of the twitter data. They exploit those in research that have been proven most

beneficial in irony detection. This encompasses features such as Hashtag Presence/Count, Mention Count, Link Presence/Count, Intensifiers & Overstatement Words Count, Emoji Presence and Has Quote. They are collected into a full feature vector, with the presence attributes as a binary value and the counts as an integer of the number of occurrences of the feature.

Affective features are those segments of a statement that are able to capture and express the sentiment and emotional intent of the speaker. These features are proven to be an important asset in irony detection (Farías et al., 2016) as irony is expressed by sentiment itself. The authors use several other systems such as AFINN, Emolex, EmoSenticNet, Linguistic Inquiry and Word Count, Dictionary and Affect in Language, and Emoji Sentiment Ranking. These systems are implemented to create a secondary feature vector that is an extraction of the multiple facets of sentiment polarity in the tweets. The feature vector is built from the scores of the applications of each system.

The model was setup as a supervised system using a SVM classifier with a radial basis function kernel. The input two feature vectors were fed into the model, along with the tweet text itself. This system was used for both SubTask A and B for the identification of ironic and non-ironic tweets. In the case of SubTask B, the authors further process the results through the model to classify as irony by polarity contrast or other irony. The last step is to then classify the remaining pool as other verbal irony or situation irony in another pass through the model. This pipeline approach allows each phase of the model to more easily converge during training as it only needs to learn two potential candidates at each step instead of four. The phases themselves are set by the difficulty of the task, with ironic and non-ironic identification as the easiest and other verbal and situational as the hardest.

The authors extraction of features and pipelined method allow them to achieve fair results on the dataset. On SubTask A and B they achieve F1 scores of 0.6216 and 0.4131. The resulting scores are similar to the previous feature based approach, although the SubTask B result is significantly lower. The authors state that their model found the greatest difficulty in the last stage of the identification process. In a strict one or the other classification like theirs, it is difficult to differentiate situational irony from other irony. This is due to the system having a lack of background knowledge or common sense needed to understand the situation. The situation is worsened when the required background is hidden by a URL or demonstrated by a removed "#not" in the dataset.

The efforts of this approach provide a great deal of insight on strategies for irony classification as well as how to avoid classification issues. While our proposed model does not extract any affective features, we can apply their structural and stylistic feature categories to our text preprocessing. This additional preprocessing would also help with the classification issue they experienced as the tweet data will contain more explicit artifacts to present hidden background knowledge. Therefore we could also implement their pipeline setup to aid in training convergence.

3 Text Processor

We developed our own text processing pipeline in order to best utilize all the information artifacts present in each tweet. As irony detection is largely expressed though sentiment and emotion, we leveraged previous works in sentiment analysis (Tang et al., 2014) and twitter analysis (Baziotis et al., 2018). We then added our own novel improvements to those strategies. We not only extract information from the tokens of the text, but also extrapolate in expanding emotions. Powerful normalization techniques are additionally applied for the easier understanding of emotions by the model.

Since Twitter is a social media network, it suffers from a large slang problem. At this current time in online platforms, users make high use of slang terms to save time when typing. This is exceptionally prevalent in Twitter which has a character limit of 140, requiring the use of space saving terms and acronyms. Additionally since human emotions cannot be expressed through typed text, emojis are commonly used to display the equivalent of human facial expressions and feelings.

In our text processing, we attempt to leverage these facts to create an input that can present and highlight possible irony. We both make the emotions intuitive and explicit while also filtering and cleaning the text input to resemble proper English, without slang and twitter formatting styles. This will result in sentences that are easier to compre-

hend, as slang such as “k”, “ok”, and “kk” all with identical meaning have different forms, will cause the model to have trouble in converging.

3.1 Dataset

The dataset for this task is that as defined by the SemEval2018 task. It consists of a corpus of 3,000 English tweets that contained the hashtags #irony, #sarcasm and #not, collected between 01/12/2014 and 04/01/2015. They were manually annotated by irony type for a total of 1,728, 267, and 401 instances of each of the respective types of irony. The remaining 604 tweets were non-ironic and are supplemented with an additional 1,792 random tweets to balance the ironic and literal instances for a total of 4,792 tweets. They are split into a training set of 80% and testing set of 20%.

We also supplement our model with an additional set of tweets for supervised pre-training. We collected an additional set of 15,000 tweets between 03/02/2019 and 03/15/2019 through the Twitter API. This set consists of 10,000 random tweets and 5,000 tweets collected with the same three hashtags. We coarsely annotate this set for binary classification using the hashtags to indicate irony. Due to the random nature of Twitter users, with a coarse automated annotation it is possible for a tweet to have the irony hashtags but not contain a truly ironic message. Since this dataset is supplementary only for pre-training, the model will still converge during standard training. Incorrect labels can be ignored as they will be a small outlying subset.

3.2 Pipeline Method

The text processing is completed over a seven step pipeline. Each step focuses on fixing or extracting a different aspect of the input. At the completion of the processing, we are presented with an input such as the example tweet in Table 2.

3.2.1 Slang and Abbreviations

The largest issue with any text from twitter is the high amount of slang, poor English and abbreviations used in each sentence. For models to converge more easily or for any model pre-trained on proper English, we need the slang fixed with proper words and the abbreviations expanded. To accomplish this we use a set of two dictionaries, one sourced from Webopedia for online chat abbreviations (Beal, 2018) which we also modified, and one that we custom wrote for slang. While writing our dictionary, we evaluated 5k tweets, searching for commonly used slang and adding translations as necessary. Multiple spellings of each slang word or phrase were also considered. The abbreviation dictionary was extended as new abbreviations were discovered. We also pruned obscure abbreviations that caused conflicts due to also representing a real word, for example $BAG \rightarrow Busting\ a\ Gut$. While writing translations, we also accounted for terms that have evolved new meanings in current slang, such as $LoL \rightarrow Laugh \not\rightarrow Laugh\ out\ Loud$ so we can convey the correct sentiment to the model.

3.2.2 Emoji Translation

Emojis and text emoticons are a very important artifact to consider when attempting to determine a sentiment as fickle as irony. The small non-linguistic cues that each emoji carries and expresses could be the difference between a sentence being determined as ironic or not (González-Ibáñez et al., 2011). Due to their widespread use in twitter, they must be considered as part of the input. Due to their unicode representation, they must be processed into a useful form for the model use. Most solutions translate them to the alias as defined by the unicode consortium (Van Hee, 2017; González et al., 2018), such as “😂→ :face-with-tears-of-joy:”. While this does allow the emoji to remain an artifact in the input, it is noisy and misleading. The key terms in that translation are *joy* (happiness) and *tears* (sadness), while the emoji is traditionally used to instead convey *laughter* in current day phrasing. This will lead to the model to not only miss the artifact that the emoji represents, but in fact misinterpret it and may cause an incorrect sentiment assumption as a result.

To resolve these poor alias translations, we create our own custom emoji alias dictionary. This dictionary is used to normalize, categorize and translate emojis to their true usage and meaning. Emojis of different forms that all express the same emotion are translated to the same catch-all word such as “love” or “Laughter”. The emojis that do not represent emotion but objects are categorized into terms such as “Food” or “Animal”.

The normalization is a key factor in this translation as there are multiple emojis that all represent the same sentiment. For example there are 19 different versions of the heart emoji, each with their own unique standard alias. Users will then often use a random combination of these all to-

Original	That was a v memorable new years kiss, lol 💋❤#lovemyrelationship | https://t.co/d1OyKPzWwA
Processed	that was a very memorable new years kiss , laughing <emotion> love </emotion> <emotion> love </emotion> <hashtag> love my relationship </hashtag> <url>

Table 2: Preprocessing pipeline transformation to benefit RNN learning

gether in a row due to the aesthetics of the different colours. The resulting standard translation would confuse any model due to the long alias names and complete random orientation and selection each instance they are encountered. This system will allow for the system to not only understand the emotions better but converge faster due to the normalization.

3.2.3 Swearing Normalization

A previously unexplored artifact in many solutions to sentiment and irony detection are swear words. Swearing is a social construct that has become very public and commonplace in the dialect of current society. It is also more than a taboo set of words, they are words that possess *emotive meaning*, expressing the speaker's state of mind (Ljung, 2010), much like other emoticons. Therefore swearing should be extracted from the source text and given importance rather then being treated as simply another input token. We normalize all swear words in our input with a <swear> token to allow the model to learn the feature and its usage.

3.2.4 Normalization and Tokenization

The input is also run through the Ekphrasis tokenizer (Baziotis et al., 2017). Ekphrasis is a preprocessor dedicated to twitter datasets and includes slang correction, emoticon translation, segmentation of hashtags, lowercasing, and handling of elongation. Also included is normalization of twitter artifacts such as URLs, usernames, and hashtags.

4 Our Approach

4.1 Overview

The approach taken in creating our model contained that task of integrating three important elements together. This included a pre-training stage and then a multi-step overall neural model. The segments of the architecture of our model begin with GloVe word embeddings that feed into the embedding matrix and embedding layers. Then we implement a supervised pre-training model using GloVe, and a supervised LSTM model. The pre-training model initially takes in a smaller dataset, using the GloVe word embeddings, and the embedding matrix to set the appropriat weights accordingly. Past this the model flows through the next layers into the 2-layer stacked LSTM model, and is trained on the irony datasets for Subtask A (binary classification) and Subtask B (multi-classification). The complete architecture of the model can be seen in the flowchart displayed in Figure 1.

4.2 Pre-Training

Since the input dataset has been heavily preprocessed to extract information, we elect to pre-train our model to gain knowledge on the form of the input text. This initial insight stored in the weights will then allow the model to converge faster on the tougher multi-class dataset as it has less it must learn. Our model consists of pre-training with our subset of additional coarsely annotated data loaded into Tensorflow, which contains word embedding weights. From this we design our model to exit the pre-training stage directly into standard model training and testing stages with the competition dataset. The tokenizer configuration with GloVe for the word embeddings form the input vectors into an embedding matrix with weights. The full pre-training model architecture is fairly simple, primarily consisting of a GloVe model with one dropout layer to prevent over fitting and a one-dimensional convolution neural network layer. This is then followed by a one-dimensional max pooling layer, and finally a dense layer. We project that with the addition of the pre-training model to the dense LSTM model we will significantly improve results due to the reduction of the initial learning hurdle of our text representation.

4.3 Dense LSTM

Once the model completes the pre-training step, it proceeds straight into the supervised LSTM layer

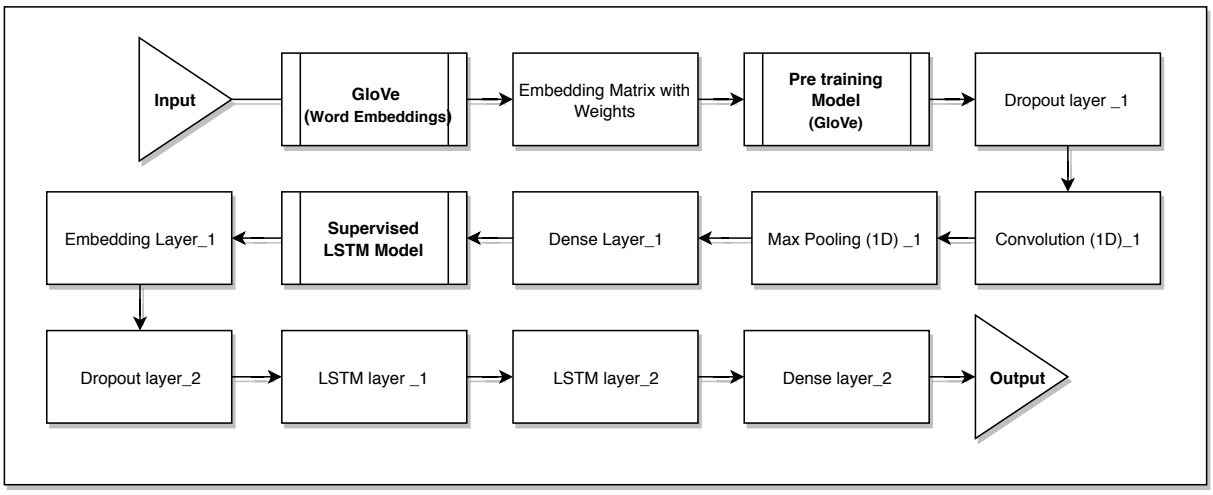

Figure 1: The architecture of our pre-trained dense LSTM model

of the model. Our model data is loaded into Keras, using the back end Tensorflow, which uses the word embedding weights from the GloVe pre-trained model. From this we designed our dense LSTM layers to use these initial weights and further fine tune them by applying supervised learning in the training and testing stages. We attempted different approaches to be able to develop the best potential model configuration using a 2-layer stacked recursive neural network with LSTM. To fine tune this model for either specific Task A or B, we modify the training process of the implementation to expect a binary or multi-class label output potential for each input tweet. The data was split into 80% train and 20% test for validation of the model, as determined by the competition. The dense LSTM model consists specifically of one embedding layer, a two layer stacked LSTM network, one dense layer, and finally one dropout layer to prevent over fitting.

4.4 Other models

While designing our full model, we experimented with other entity layers, which were not able to achieve results of the same quality as our final design. We attempted to use a 3-layer RNN (Recursive Neural Network) with a combination of our pre-training layers using GLoVe for the word embeddings. This model was not able to perform similar to the quality of the dense LSTM. We believe this because using too many layers of RNNs in a model can cause shortage in short term memory due to the vanishing gradient problem (Hochreiter, 1998). When gradient value becomes extremely small during backpropagation, it does not contribute to the learning through weight adjustments. This will cause features in the earlier RNN layers to not be captured and evaluated. Overall, by switching to the current model of using LSTM instead, allowed us more control and variability with embedding weights and therefore better results.

5 Experiment Set-Up

5.1 Baseline System

We experiment on not only our own model, but also a baseline system to provide context to our results. The SemEval task creators provide a simple baseline system as a minimum score threshold for potential results. It is based on simple token unigram features and provides predictions for both Subtask A and Subtask B. We run this model using the unprocessed original dataset. Therefore obtaining a minimum F1 score context to compare the results of our model and pre-processing to.

We additionally implemented a secondary baseline system due to recent advances in NLP. The Bidirectional Encoder Representations from Transformers, or BERT model is the current top model for any NLP tasks (Devlin et al., 2018). BERT is a pre-trained model trained on a large English corpus, where it is then fine tuned for an exact prediction task. We use the released `BERT-Base, Uncased` model as our secondary baseline. We select the base model due to training time, as we do not possess the computing power to fine tune the large model, even though it would produce better results. The uncased model is used due to our pre-processing to-

kenizer removing the case of all letters.

5.2 Data

The dataset used for the experiment is that discussed in Section 3.1. For the training of our model, we use both our supplemental pre-training data and the supplied irony dataset for traditional supervised training. When training the BERT baseline, we use only the the supplied irony dataset due to the model already being pre-trained on English.

5.3 Training Details

5.3.1 Our Model

We ran our model on Keras, using the back-end of TensorFlow, through Anaconda Python 3.6.6. We scaled the training using Google Cloud GPU to run the full datasets, of the model instead of running on the lower powered Anaconda CPU mode. The model was trained with a total of 15 epochs, using a sequential model with Adam as the optimizer, and using an activation function of softmax. The maximum sequence length produced by the tokenizer was 128, to fit the full processed tokenized tweets. Due to limited GPU and CPU space the model is not able to handle large sequence lengths compared to other common models, which can handle up to 400 length sequences.

5.3.2 BERT

The BERT-Base model is loaded from a set of TensorFlow checkpoints which contain all the pre-trained weights. To fine-tune the model for either Subtask A or B, we use a modified version of the training process for the Cola dataset, as provided by the BERT authors. This implements a fine-tuning process for text classification for either binary or multi-class, depending on the Subtask. We reformat the data into the format that the BERT classifier expects, splitting the labelled training data into 90% train and 10% development. Then the testing data is prepared without labels.

We run the BERT classifier fine-tuning on in TensorFlow CPU mode on Anaconda Python 3.7, as we do not have a GPU and CUDA development environment. The default BERT tokenizer and configuration is used to prepare the input vector as we intend this experiment to provide a baseline performance. We train the model for a total of 3 epochs, using a learning rate of 2×10^{-5}. The maximum sequence length produced by the tokenizer is 400, to fit the full processed tokenized tweet and we batch these input messages in sets of 8 for training.

6 Results

6.1 Results on the Pre-Processed Dataset

We first conduct our experiment as described on three models. We use the basic supervised LSTM layers from our model, our full pre-trained model and BERT as our test cases. We present the results of these tests in Table 3. Also included is the baseline unigram model from the SemEval competition, along with the models of several high ranking competitors.

Our strictly supervised LSTM test achieves poor results, under-performing the baseline in Task A and scoring only slightly higher in Task B. Our dense LSTM model though performs very well, achieving an F1 score of 0.6500 and 0.5076 in the respective tasks. This ranks us above the previous #3 competitor in the competition for Task A and as the new #1 score for Task B. This proves our model to be quite effective as compared to the previous Task B winner (Ghosh and Veale, 2018), we are able to slightly improve upon the score of that more difficult and marginally improve the Task A score by an extra 0.1254 points at the same time. Although the Dense LSTM model (Wu et al., 2018) arguably rivals ours overall as it provides a 7.7% increase in Task A at a cost of only 2.5% decrease in Task B's results.

The BERT Base model provides similar but improved results to our dense LSTM model. In Task A, we see a marginal 0.0011 increase in F1 score, but a much improved 0.207 increase in Task B. Our official rankings do not change in the case of the BERT model, but we do now achieve a more notable Task B first place, as previously there was only a 0.0002 score difference between our model and the Siamese LSTM model. These rankings also further validate the success of our model as we were almost able to match the performance of BERT. Due to the nature of the transformer architecture of BERT, it should easily bypass the results of any LSTM network and to achieve similar results indicates our model is performing at a very high level. It is important to note that BERT under normal operation should expect to achieve better results then we present here. Due to our lack of computing power and CPU training restriction, we were only able to train the smaller BERT model for a short number of epochs. With more power

System	Subtask A			Subtask B		
	Precision	Test	Rank	Precision	Test	Rank
Supervised LSTM	0.6372	0.6085	#15	0.5498	0.4087	#10
GloVe + Pre-training + Supervised LSTM	0.6831	0.6500	#3	0.6777	0.5076	#1
BERT Base	0.6823	0.6511	#3	0.6771	**0.5283**	#1
Densely Connected LSTM (Wu et al., 2018)	0.6304	**0.7054**	#1	—	0.4947	#3
LSTM Siamese Network (Ghosh and Veale, 2018)	0.6449	0.5257	#30	0.5768	0.5074	#1
Affective Content (Pamungkas and Patti, 2018)	0.6790	0.6216	#10	0.5550	0.4131	#9
Baseline Unigram	—	0.6263	—	—	0.3470	—

Table 3: Results of the classification on the pre-processed irony dataset

and training time, the model will be able to better capture the features of the data. Nevertheless, in its underpowered capacity it is still a high powerful model worth comparing to.

6.2 Feature Ablation

In order to evaluate our secondary contribution of a custom preprocessing pipeline, we perform a feature ablation on the differing artifacts extracted and cleaned in our pipeline. We establish three main test cases for this evaluation. First we use the standard baseline NLP normalization tactics where we normalize features such as URL's and names to token form, as well as translating emoji's to their alias description. We also use the standard un-modified slang correction dictionary bundled with the ekphrasis tool to compare the power of our custom translation. For our second test, we apply the same normalization but instead of the previous emoji and slang corrections, implement our custom normalizing dictionaries. For the final test we use the full preprocessing pipline described in Section 3. This feature ablation is evaluated with our full dense LSTM model and with BERT on the more difficult Task B, with the results displayed in Table 4.

The results of this test demonstrate a linear increase in F1 score as we increase features. Between the baseline and our custom translation dictionaries, we see a 5.3% and 0.8% respective increase in F1 score in the LSTM and BERT models. Then implementing the full pipeline provides another 5.9% and 3.5% respective increase.

In the case of the LSTM model, the slang correction and emotion normalization and preservation that our custom dictionaries provide achieved a significant boost, while in the case of BERT was close to a non-factor. This is likely due to the tokenization method of BERT which causes strange splitting of our token artifacts. For example the token `<emoji>` is tokenized into five separate `'<', 'em', '##oj', '##i', '>'` input tokens. This in turn will impair learning in the model as it can no longer extract the information that the token provides, it must first learn to reconstruct it. Our custom dictionaries are designed to introduce a large number of these tokens to aid in the identification of emotions in a standard LSTM model. This benefit therefore does not translate over to BERT and should be reconstructed for use in further BERT models.

As expected, the remaining segments of our preprocessing pipeline provided an equivalent secondary boost due to the cleaning and tokenization provided. Again we see a lower benefit in BERT due to the introduction of the `<swear>` tokens, but still a positive benefit overall. Our preprocessing provides a comprehensive 11.9% total increase in F1 score in the LSTM model. Therefore we can verify that our lengthy and custom preprocessing pipeline is successful and a tool worth refining, extending and using.

7 Future Work

7.1 Text Processing

The results in our feature ablation in Section 6.2 show the custom pre-processing to increase F1 score. Therefore we propose to improve the text processing to potentially improve our F1 score results further. The first major improvement would be to crowd-scource a larger custom slang and emoji dictionary. In order to ensure the dictionary covers 90% of slang cases, we would need to extract a large twitter dataset with emojis and have the tweets evaluated for slang and emoji translations. Since slang and internet speak constantly evolve and there is no upper-limit to misspellings and shortcuts, a large dictionary would be required

Feature	Pre-Trained LSTM		BERT Base	
	Dev	Test	Dev	Test
Baseline Normalization	0.6593	0.4538	0.6796	0.5066
Custom Dictionaries	0.6532	0.4791	0.6749	0.5105
Full Pipeline	0.6777	0.5076	0.6771	0.5283

Table 4: Feature ablation results on the multi-class irony classification

for pre-processing on larger scale datasets

On the linguistic side of the text processing, we also propose an improvement to the swearing normalization. When a human uses a swear word, they select one of the correct intensity to express their sentiment. Therefore we propose a normalization scheme where we will add a quantitative scale component to our tokens. To determine the values to apply, we must analyze the intensity of swear words over a sample corpus and then rate them for translation. Therefore the model will have an accurate gauge of the negative sentiment from the words.

7.2 Model Improvements

While the model performed well, there are still improvements that could be made to the model architecture to improve overall efficiency and performance. As of now, our model was not able to process large datasets due to configuration and our available computing power. Therefore we propose to execute the the same model on a larger scale with additional training time, length and epochs. With additional GPU space to run these dense models we predict we can further achieve better precision and F1 scores. With regards to the model architecture, we can further optimize it through additional deepening of the model, by experimenting with adding more layer combinations to the stacked LSTM model. This may help improve the model even more and can even further improve the algorithm due to the ability of each layer to learn a new feature of the input.

7.3 BERT Integration

Based on the results of both models individually, we propose exploring a combination of our Dense LSTM integrated with a BERT layer for additional classification. We suggest a smooth model where data will flow from the word embeddings of the GLoVe pre-training model to the LSTM model layer. The predictions will then finally integrate into a BERT based layer to further classify the data for the Subtask A, and Subtask B datsets. This proposed model will present unique challenges largely due to the amount of computing power required to fine tune BERT alone. It will be necessary to possess enough computing resources to run this heavy dense algorithm. Through this integration of the BERT base model we expect our model results to increase due to the additional layer of classification.

7.4 Conclusion

We introduced Dense LSTM, a pre-trained LSTM network that is capable of separating ironic statements from non-ironic. Additionally it can classify those ironic statements at a fine grained level with high accuracy into specific types of irony. The system is able to perform exceptionally at the classification of types of irony but can still be improved in the binary classification regard.

References

Christos Baziotis, Nikos Athanasiou, Alexandra Chronopoulou, Athanasia Kolovou, Georgios Paraskevopoulos, Nikolaos Ellinas, Shrikanth Narayanan, and Alexandros Potamianos. 2018. Ntua-slp at semeval-2018 task 1: predicting affective content in tweets with deep attentive rnns and transfer learning. *arXiv preprint arXiv:1804.06658*.

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada. Association for Computational Linguistics.

Vangie Beal. 2018. Huge list of texting and online chat abbreviations.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Eslefeve. 2018. Semeval-2018 task 3 - irony detection in english tweets.

Delia Irazú Hernańdez Farías, Viviana Patti, and Paolo Rosso. 2016. Irony detection in twitter: The role of affective content. *ACM Transactions on Internet Technology (TOIT)*, 16(3):19.

Aniruddha Ghosh and Tony Veale. 2018. Ironymagnet at semeval-2018 task 3: A siamese network for irony detection in social media. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 570–575, New Orleans, Louisiana. Association for Computational Linguistics.

José-Ángel González, Lluís-F Hurtado, and Ferran Pla. 2018. Elirf-upv at semeval-2018 tasks 1 and 3: Affect and irony detection in tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 565–569.

Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. 2011. Identifying sarcasm in twitter: A closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 581–586, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Chenghua Lin and Yulan He. 2009. Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 375–384. ACM.

Magnus Ljung. 2010. *Swearing: A cross-cultural linguistic study*. Springer.

Edison Marrese-Taylor, Suzana Ilic, Jorge A Balazs, Yutaka Matsuo, and Helmut Prendinger. 2018. Iiidyt at semeval-2018 task 3: Irony detection in english tweets. *arXiv preprint arXiv:1804.08094*.

Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. 2007. Topic sentiment mixture: modeling facets and opinions in weblogs. In *Proceedings of the 16th international conference on World Wide Web*, pages 171–180. ACM.

Elena Mikhalkova and Yuri Karyakin. 2017. Detecting intentional lexical ambiguity in english puns. *arXiv preprint arXiv:1707.05468*.

Elena Mikhalkova, Yuri Karyakin, Alexander Voronov, Dmitry Grigoriev, and Artem Leoznov. 2018. Punfields at semeval-2018 task 3: Detecting irony by tools of humor analysis. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 541–545.

Debora Nozza, Elisabetta Fersini, and Enza Messina. 2016. Unsupervised irony detection: A probabilistic model with word embeddings. In *KDIR*, pages 68–76.

Endang Wahyu Pamungkas and Viviana Patti. 2018. # nondicevosulserio at semeval-2018 task 3: Exploiting emojis and affective content for irony detection in english tweets. In *International Workshop on Semantic Evaluation*, pages 649–654. Association for Computational Linguistics.

Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. 2014. Coooolll: A deep learning system for twitter sentiment classification. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 208–212.

Cynthia Van Hee. 2017. *Can machines sense irony? : exploring automatic irony detection on social media*. Ph.D. thesis, Ghent University.

Cynthia Van Hee, Els Lefever, and Véronique Hoste. 2018. Semeval-2018 task 3: Irony detection in english tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 39–50.

Chuhan Wu, Fangzhao Wu, Sixing Wu, Junxin Liu, Zhigang Yuan, and Yongfeng Huang. 2018. Thu_ngn at semeval-2018 task 3: Tweet irony detection with densely connected lstm and multi-task learning. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 51–56, New Orleans, Louisiana. Association for Computational Linguistics.