CSC413 Final Project Report: Sarcasm Detection

Jonathan Chen **Jiongan Mu** **Rohit Shetty** **Mohit Bawa**

1 Motivation and Problem Definition

According to linguist Robert Gibbs, sarcasm is the phenomenon where “words [are] used to express something other than and especially the opposite of the literal meaning of a sentence" [1]. Detecting sarcasm has proven to be a challenging task for both humans and Natural Language Processing (NLP) models as sarcasm’s ironic nature often depends on a variety of factors that include sentence content, conversational context, tone and non-verbal cues.

With this in mind, we are aiming to explore Neural Network (NN) based sarcasm detection models that can be integrated with pre-existing NLP applications, making sentiment analysis more powerful, expressive and accurate in the process. We want to focus on determining the relationship between sentence content and sarcasm, exploring the lexical, syntactical and semantic features that define a sarcastic sentence and learning these features with NN/DL algorithms and models. Finally, we aim to explore if the NN/DL models we develop can exceed human performance in sarcasm detection.

2 Summary of Prior Work

Over the years, the field of sarcasm detection has steadily grown and matured. Our project is built off of the work of a variety of research topics under the broader sarcasm detection field.

The work of Porwal et al. (2018) [2], which explored the effectiveness of using a Long Short-Term Memory (LSTM) network, a type of Recurrent Neural Network (RNN), for Twitter sarcasm detection heavily inspired one of our proposed RNN-based model architectures.

In addition, the work of Ghosh et al. (2017) [3] was especially relevant in identifying a baseline model to benchmark our models’ performance. Focusing on both single and multi-sentence level context (or "conversational context"), Ghosh et al. [3] developed a SVM (Support Vector Machine) with discrete features to detect sarcasm. They trained and tested their model using the Sarcasm Corpus v2 dataset, which we have also used for our implemented models.

Research from Xu et al. (2019) [4] inspired the idea to compare the effectiveness of LSTM networks against Transformers in sarcasm detection. Extending on the BERT-based classifier model from Huggingface that Xu et al. [4] developed, we have explored sarcasm detection with the RoBERTa model, an optimized variant of the BERT model. This paper also has the added benefit of using the Sarcasm Corpus v2 dataset as well, so evaluating performance across multiple models will be straightforward.

Lastly, we’ll compare our results with the work of Gole et al. (2023) [5] which explores the effectiveness of twelve different OpenAI GPT-based models on sarcasm detection. This study uses a portion of the SARC dataset which we have access to as well.

3 Dataset

Because there has been substantial research completed in the sarcasm detection field, there are many publicly available datasets. We have concentrated primarily on two of these publicly available datasets: **Sarcasm Corpus v2 dataset**, created by Oraby et al [6], and **SARC** (Sarcasm on Reddit),

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

created by Khodak et al [7]. Both of these datasets have a balanced distribution (50/50) of sarcastic and non-sarcastic examples.

While Porwal et al's [2] LSTM network was focused on sarcasm in the context of tweets from Twitter, we wanted to investigate sarcasm on multiple social media platforms for better generalization. The **Sarcasm Corpus v2** dataset is comprised of various online discussion forum posts that are labelled as sarcastic or not sarcastic through crowd-sourced annotations. For our project we have used the GEN (general sarcasm) corpus, containing 6520 examples. The **SARC** dataset is comprised of 1010826 Reddit comments, self-annotated as sarcastic/not sarcastic by the author of the comment. For our project, we have decided to truncate this dataset to just 50000 examples (while still maintaining the equal distribution of sarcastic/non-sarcastic examples) due to our limited compute resources. Each dataset was then divided into a split of 60% training set, 20% test set, and 20% validation set.

For model training and validation, we merged the training and validation sets from both Sarcasm Corpus v2 and SARC to create a combined training set and a combined validation set (see data pre-processing steps for more detail on this process).

However, during test time, we used the Sarcasm Corpus v2 and SARC test sets separately so that we could compare our model's performance with the various baseline models mentioned above.

3.1 Data Pre-Processing

Reading the data: Both the GEN corpus and SARC datasets were downloaded from Kaggle as csv files. Using the gdown and pandas libraries, each csv file was read into it's own independent dataframe for initial processing.

Processing the data: With the dataset stored in memory as dataframes, unnecessary columns were dropped, columns were renamed/re-ordered and class labels for sarcastic and not sarcastic were modified to 1 and 0 respectively. The final processed dataframe comprised of 2 columns (text example, label).

Splitting and Combining Datasets: As mentioned earlier both datasets were split with a 60% training set, 20% test set, and 20% validation set distribution. The 2 training sets were then concatenated and shuffled randomly before being converted into Python lists, where each element is a tuple (text example, label). A similar process was followed for combining the validation sets together. However, the test sets were kept separate and converted to lists independently.

Note: All code associated with the data pre-processing step can be found under the "Data Pre-Processing" section in any of the submitted notebooks.

4 Project Details

4.1 Formulating Sarcasm Detection as an ML Problem

Given that our project focused on a Natural Language Proccessing (NLP) classification task (sarcasm detection), we needed a mathematical representation of English words for our models to take in as input. In particular we needed a method to map a single English word to specific indices, allowing us to tokenize our input data. We approached single-word tokenization in two distinct ways, building our own custom tokenization as well as using pre-trained tokenization from the GLoVE representation.

Custom Tokenization: Using the torchtext library we assign each unique word in our training set to a corresponding index, building a resulting Vocab object. In addition, similar to Lab 7, we reserved 4 special tokens to represent placeholder values. The four values include:

- <bos>: represents the beginning of a sequence
- <eos>: represents the end of a sequence
- <unk>: generic token for an word not in Vocab
- <pad>: padding token to make sure all sequences in a batch have same length

GloVe Tokenization: We used the GloVe representation available from the torchtext library. We specified to use the pre-trained GloVe representation trained on 6 billion input tokens with an embedding size of 300.

Note: See the "Tokenization" section in the "Vanilla_RNN" or "GLoVE_RNN" notebook for more details

In addition, given that a sentence is a sequence of words, detecting sarcasm within a given sentence requires finding an NN model that is able to handle inputs of various length. This makes the RNN and Transformer architectures especially relevant for our use case. We also formulated sarcasm detection as a binary classification problem, which helped to inform our choices of activation functions and loss functions.

RNN Architectures Explored: We decided to explore 6 different RNN architectures, varying the cell type of the RNN and the tokenization method used.

- **vanillaRNN**: This neural network is made up of a word embedding layer, 1 bi-directional RNN layer and a final classifier layer.
- **vanillaLSTM**: This neural network is made up of a word embedding layer, 1 bi-directional LSTM layer and a final classifier layer.
- **vanillaGRU**: This neural network is made up of a word embedding layer, 1 bi-directional GRU layer and a final classifier layer.
- **gloveRNN**: This neural network is made up of a word embedding layer, 1 bi-directional RNN layer and a final classifier layer. However, the word embedding weights come from the pre-trained GloVe embedding layer and are frozen during training of our model. In other words, we have applied transfer learning .
- **gloveLSTM**: This neural network is made up of a word embedding layer, 1 bi-directional LSTM layer and a final classifier layer. Similar to the gloveRNN model, we have used transfer learning for the embedding layer.
- **gloveGRU**: This neural network is made up of a word embedding layer, 1 bi-directional GRU layer and a final classifier layer. Similar to the gloveRNN model, we have used transfer learning for the embedding layer.

Note: See the "Vanilla_RNN" notebook for the code implementing the various vanilla RNN-based models and "GLoVE_RNN" for code related to GloVe RNN-based models.

Since sarcasm detection is a binary classification problem, we chose to use the sigmoid activation function along with the Binary Cross Entropy loss function for all the RNN architectures explored above.

In addition, we choose an embedding size of 300 for all RNN-based models described above not only because 300 is commonly used in GloVe embeddings, but it can also effectively capture semantic information about words, which is especially important in our task of sarcasm detection. An embedding size of 300 should be robust enough to encapsulate sarcasm while still ensuring computational efficiency under the limited computing resources that we have.

Finally, we used a bi-directional RNN layer for each model in order to better capture contextual dependencies in the our text examples.

Transformer Architectures Explored: For Transformer-based models, we use the pre-trained RoBERTa model. Roberta is a bidirectional transformer architecture, meaning it captures context from both directions of a sequence. This enables it to understand and classify sequences of text. We experimented with various configurations during training.

- **Fine-tuned model**: We load the pre-trained weights and freeze all parameters except those in the final classification layer. We then fine tune the model on the training set.
- **Standard model**: Again, the pre-trained weights are loaded but simply used as an initialization. The final classification layer is randomly initialized. Then the entire model is trained on the training set.

- **Weighted Loss Function Variant**: We use the same process as the standard model but with a custom loss function that assigns higher weights to sarcastic instances. This is used to make the model more sensitive to sarcasm.

4.2 Experiment Methodology

Grid Search for RNN's: We use grid search to find the best combination of hyperparameters for all of our RNN-based models (both banilla and GloVe RNN's, LSTM's, and GRU's). We vary two hyperparameters: batch size and learning rate while fixing the number of epochs to 2 due to limited computation resources. This gives each configuration a fair chance to learn from the data. For each model, the combination with the highest validation accuracy is selected as the candidate model to be used for evaluation.

Once we have the best hyperparameter combinations, we train these candidate models based on these hyperparameters, using 10 epochs instead for more robust performance. Several evaluation metrics, including accuracy, precision, recall are used to evaluate the performance of the models.

Here are the best hyperparameter configuration for each RNN model:

- **Vanilla RNN**: batch size: 500, learning rate: 0.1
- **Vanilla LSTM**: batch size: 1000, learning rate: 0.01
- **Vanilla GRU**: batch size: 1000, learning rate: 0.05
- **GloVe RNN**: batch size: 1000, learning rate: 0.01
- **GloVe LSTM**: batch size: 500, learning rate: 0.01
- **GloVe GRU**: batch size: 500, learning rate: 0.01

Hyperparmaeter Tunring for Transformers: For transformers, we used the recommended hyperparameters from Roberta's documentation and tweaked them manually to obtain a the highest possible validation accuracy without causing performance issues during training. The batch size was limited to 32 to avoid running out of GPU memory. We train each model for 2 epochs which is enough to achieve reasonable performance.

Here are the best configurations for each transformer model:

- **Fine-tuned model**: batch size 32, learning rate: 4e-06
- **Standard model**: batch size 32, learning rate: 5e-05
- **Weighted Loss Function Variant**: batch size 32, learning rate: 5e-05

4.3 Evaluation of Model

In order to evaluate the performance of our model, we have chosen 3 metrics to consider: Precision, Recall and Accuracy

We calculated the precision and recall for both the sarcastic class and non-sarcastic class. Note: the code for these computations can be found in the "Utils" section in the"GLoVE_RNN" and "Vanilla_RNN" notebooks as well as the "print_metrics" function in the "Transformer" notebook.

- **Precision for Sarcastic Class (`precision_sarcastic`):**
  - **True Positives (`true_pos`):** the number of instances where both the prediction (`y`) and the label (`t`) are 1 (sarcastic).
  - **Total Predicted Positives (`total_pred_pos`):** the number of instances where the model predicts 1.
  - **Calculation:** `precision_sarcastic = true_pos / total_pred_pos`
- **Precision for Non-Sarcastic Class (`precision_nonsarcastic`):**
  - **True negatives (`true_neg`):** the number of instances where both the prediction and the label are 0 (non-sarcastic).

  - **Total Predicted Negatives** (`total_pred_neg`)**:** the number of instances where the model predicts 0.
  - **Calculation:** `precision_nonsarcastic = true_neg / total_pred_neg`

- **Recall for Sarcastic Class** (`recall_sarcastic`)**:**
  - **True Positives** (`true_pos`)**:** Same as above.
  - **Total Actual Positives** (`total_actual_pos`)**:** the number of instances where the label is 1.
  - **Calculation:** `recall_sarcastic = true_pos / total_actual_pos`
- **Recall for Non-Sarcastic Class** (`recall_nonsarcastic`)**:**
  - **True Negatives** (`true_neg`)**:** Same as above
  - **Total Actual Negatives** (`total_actual_neg`)**:** the number of instances where the label is 0.
  - **Calculation:** `recall_nonsarcastic = true_neg / total_actual_neg`

With the experiment methodology described above, here were the empirical results gathered for the Sarcasm Corpus (discussion form) dataset.

Table 1: Performance on Sarcasm Corpus dataset

Model	Sarcastic		Non-sarcastic		Accuracy
	Precision	Recall	Precision	Recall	
SVM$_r$ ([Ghosh et al., 2017])	65.55	66.67	66.10	64.96	n/a
LSTM$_r$ ([Xu et al., 2019])	67.76	60.80	62.75	69.54	65.05
LSTM$_{conditional}$ ([Xu et al., 2019])	71.32	80.13	**75.82**	65.93	73.23
BERT$_r$ ([Xu et al., 2019])	68.60	**84.14**	44.94	25.15	64.10
Best Performing Vanilla RNN Model	78.05	19.97	55.00	**94.57**	56.04
Best Performing Vanilla LSTM Model	69.82	49.45	61.88	79.34	63.73
Best Performing Vanilla GRU Model	64.24	63.34	65.03	65.91	65.13
Best Performing GloVe RNN Model	77.64	39.24	59.10	88.60	63.74
Best Performing GloVe LSTM Model	72.79	61.68	66.49	76.73	67.23
Best Performing GloVe GRU Model	65.26	65.95	65.26	64.56	64.54
Best Performing Transformer Model	**83.80**	72.45	75.41	85.78	**79.06**

With the experiment methodology described above, here were the empirical results gathered for the SARC (Reddit) dataset.

Table 2: Performance on SARC dataset

Model	Sarcastic		Non-sarcastic		Accuracy
	Precision	Recall	Precision	Recall	
LSTM$_r$ ([Xu et al., 2019])	57.64	**80.95**	68.28	40.80	60.83
LSTM$_{conditional}$ ([Xu et al., 2019])	67.86	66.58	66.83	68.11	67.34
BERT$_r$ ([Xu et al., 2019])	**82.23**	78.25	**80.02**	**92.18**	76.08
GPT-3 - davinci model ([Gole et al., 2023])	n/a	n/a	n/a	n/a	**81.00**
GPT-3.5 - gpt-3.5-turbo model ([Gole et al., 2023])	n/a	n/a	n/a	n/a	77.60
Best Performing Vanilla RNN Model	68.69	38.32	56.91	82.34	61.14
Best Performing Vanilla LSTM Model	66.44	60.03	63.18	69.35	63.14
Best Performing Vanilla GRU Model	67.00	45.08	57.99	77.35	61.14
Best Performing GloVe RNN Model	63.61	59.02	61.59	66.06	62.24
Best Performing GloVe LSTM Model	64.34	69.65	66.73	61.19	63.94
Best Performing GloVe GRU Model	56.10	77.14	62.78	38.98	60.74
Best Performing Transformer Model	74.41	69.32	71.47	76.32	72.83

5 Conclusions

The task of sentiment analysis such as sarcasm detection remains a challenging task in the field of natural language processing.

By analysing the tables above, it seems that the models we trained ourselves perform slightly better on the Sarcasm Corpus dataset compared to the SARC dataset. This suggests that it's possible that the examples in the Sarcasm Corpus dataset are less diverse and easier to interpret.

Among all the RNN models we explored, the results show that an LSTM layer performs better than RNN and GRU layers. LSTM models perform consistently across both the Sarcasm Corpus and SARC datasets, and have a balanced recall and precision. GRU has a similar performance compared to LSTM, but slightly less consistent overall. Both LSTM and GRU perform better than RNN due to their architectures' ability to mitigate the vanishing gradient problem. Although the difference is not significant in our empirical results, we believe with more training data and computational resources, both LSTM and GRU architectures will perform substantially better than RNN due to their gating mechanisms, which can capture and "remember" the sequential and contextual information more effectively. We also find that GloVe-augmented models generally perform better than our Vanilla models. This can be due to the fact that we use GloVe embeddings of size 300. The GloVe representation is pre-trained and provides a rich representation of word semantics and relations, which is important for detecting underlying sarcasm and more robust than our custom trained embedding layer.

The transformer models we explored show promising results. Specifically, the pre-trained Roberta model, which we trained further for sarcasm detection, was able to outperform all other models when evaluated on the Sarcasm Corpus test set (see Table 1 above). Note that the Sarcasm Corpus test set we used may differ from the ones that other research teams used in their evaluation. The transformer model also performs well on the SARC dataset but is outperformed by the GPT models. Again, the exact specification of the test set may differ between research teams. During training, we learned that transformer models like RoBERTa require very careful tuning of hyperparameters. If the learning rate is too small or large, the gradients can explode or vanish during training which leads to errors in loss computation. We also noticed that very few epochs were needed to achieve reasonable performance. After just two epochs of training, the performance benefits of further training were marginal. One possible extension of our research would be to experiment with a learning rate that becomes smaller towards the end of training. This would allow the optimizer to precisely navigate the loss landscape during the final few epochs.

It's worth mentioning that even humans have trouble detecting sarcasm for many of the samples in our datasets. This is because sarcasm is inherently difficult to understand, especially without sufficient context. The samples in our dataset do not include context, so the model is forced to make a guess based purely on the sentence content provided in a single post. In this situation, it's reasonable to expect imperfect results. To solve this problem, we could extend our research by incorporating context in the datasets and train models that are able to process and interpret the context in addition to the sarcastic or non-sarcastic post. For example, we could look at a set of posts and their (potentially) sarcastic replies. In addition, both Ghosh et al. (2017) and Xu et al. (2019) [4] investigated developing architectures that would also factor in parent posts/replies as a proxy for prior conversational context. Building off similar work could also be another extending option to explore.

References

[1] R. W. Gibbs, "On the Psycholinguistics of Sarcasm," Journal of experimental psychology. General, vol. 115, no. 1, pp. 3–15, 1986, doi: 10.1037/0096-3445.115.1.3.

[2] S. Porwal, G. Ostwal, A. Phadtare, M. Pandey and M. V. Marathe, "Sarcasm Detection Using Recurrent Neural Network," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 746-748, doi: 10.1109/ICCONS.2018.8663147.

[3] D. Ghosh, A. R. Fabbri, and S. Muresan, "The Role of Conversation Context for Sarcasm Detection in Online Interactions," arXiv (Cornell University), 2017, doi: 10.48550/arxiv.1707.06226.

[4] L. Xu, V. Xu, "Project Report: Sarcasm Detection", CS224n: Natural Language Processing with Deep Learning (Stanford University), 2019

[5] M. Gole, W. Nwadiugwu, and A. Miranskyy, "On Sarcasm Detection with OpenAI GPT-based Models", arXiv (Cornell University), 2023, doi: 10.48550/arXiv.2312.04642

[6] S. Oraby, V. Harrison, L. Reed, E. Hernandez, E. Riloff, and M. Walker, “Creating and Characterizing a Diverse Corpus of Sarcasm in Dialogue,” 2017. doi: 10.48550/arxiv.1709.05404.

[7] M. Khodak, N. Saunshi, and K. Vodrahalli, “A Large Self-Annotated Corpus for Sarcasm,” 2017, doi: 10.48550/arxiv.1704.05579.