Sarcasm Generation using Character-Level RNNs

Seema Rida
COGS 185 Final Project

June 14, 2025

Abstract

This project investigates whether character-level RNNs can generate sarcastic headlines using only text. A vanilla RNN, LSTM, and GRU were compared on a dataset from *The Onion*, and LSTM performed best. After tuning, the model produced outputs that showed elements of sarcasm despite lacking full context.

1 Introduction

Sarcasm is an interesting mode of language that, in some ways, challenges direct language understanding. It depends on things like tone and facial expressions. However, once it's stripped of those contextual cues and you're looking at text alone, it becomes much harder to tell if something is sarcastic. This led me to my question: Can a character-level RNN generate sarcasm using nothing but sequential text?

We know that language comprehension largely depends on memory and context, and so this got me thinking that sarcasm might be one of the most context and memory dependent modes of language. For a model to be able to detect sarcasm properly, and also generate it, it would have to be able to learn patterns that hint at subtle ironic intent. So to dive deeper into this, I will be studying the basic structure of Recurrent Neural Networks (RNNs), which are models that handle sequential data, and then explore different architectures to see what other structures allow for the kind of memory I deem necessary for sarcasm.

RNNs work by updating a hidden state that carries information from previous inputs; this acts as a sort of 'memory' for the model. They do have a downside: the vanishing gradient problem. This makes it so that the model cannot learn long-term dependencies. I thought this would affect performance when dealing with the sarcasm dataset, because, as said before, sarcasm is context-dependent — it might rely on something mentioned earlier in the sentence or how it's phrased. In this project, I will be comparing the performance of a vanilla RNN with Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) in order to address the memory issues.

My goal in this project is to address sarcasm as both a cognitive and computational challenge. Of course, at an introductory level, the models I'm using lack access to rich context, but that limitation will actually aid in allowing us to isolate the role of language structure alone and see if sarcasm can be learned from textual patterns only. By exploring a variety of architectures and tuning hyperparameters, I hope to find an effective method for modeling sarcastic language with minimal context - and show that even a computer can learn a bit of wit.

2 Method

2.1 Data

- **Dataset:** I used the News Headlines Dataset for Sarcasm Detection by Rishabh Misra. It contains over 26,000 labeled headlines, with sarcastic entries sourced from *The Onion* and non-sarcastic entries from HuffPost. The Onion is a satirical news site that is known for its ironic headlines. All headlines are relatively short and professionally written.
- **Preprocessing:** I preprocessed the dataset by filtering for only sarcastic entries. I used a script called `json_to_sarcasm.txt.py` to convert the original JSON format into a plain text file.

2.2 Architectures

I will be using a character-level RNN. This means that I plan to use it to generate words and sentences character by character rather than word by word. I use a vanilla RNN, LSTM, and GRU which are all different architectures of recurrent neural networks.

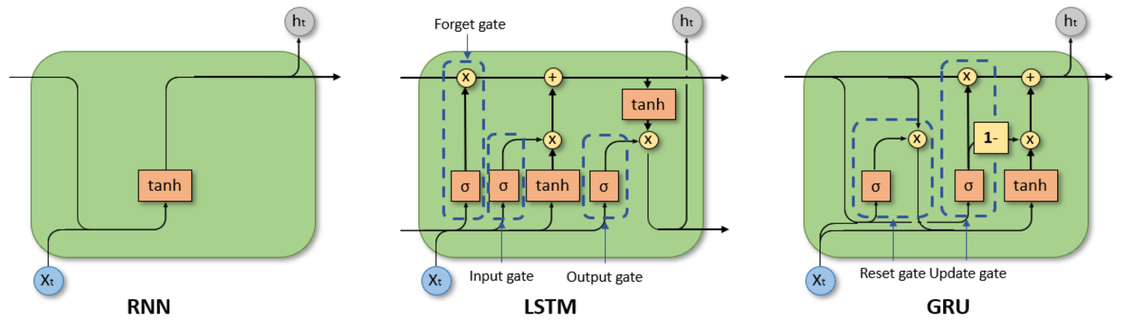

Figure 1: Comparison of RNN, LSTM, and GRU architectures. Source: Suvankar Maity on LinkedIn

- **Recurrent Neural Network (RNN):** A basic RNN updates its hidden state at each time step by combining the previous hidden state with the current input. The previous hidden state acts as the model's memory, allowing information to be carried forward and context to build over time. However, RNNs suffer from the vanishing gradient problem, meaning the model struggles to retain long-term dependencies. This makes it harder to "remember" things from earlier in a sequence. I figured this would be an issue worth investigating, especially with sarcasm, since irony often relies on context - something that might be mentioned much earlier in the sentence or subtly implied through phrasing.
- **Long Short-Term Memory (LSTM):** An LSTM addresses the vanishing gradient problem by incorporating gates and memory cells, which help control the flow of information in the network where they decide what's remembered, what's forgotten, and what's passed to the next time step. This allows for the model to memorize longer dependencies and better understand more complex sequential relationships.
- **Gated Recurrent Unit (GRU):** A GRU also addresses the vanishing gradient problem but simplifies the architecture by combining the forget and input gates into a single update gate. This makes GRUs faster to train and easier to implement.

I trained all three models to generate sarcastic headlines. In the following sections, I compare their outputs and evaluate how well each model captures the sarcastic tone and structure of the training data.

3 Experiments and Results

3.1 Model Comparison

3.1.1 Training and Generation

First, I trained the vanilla RNN, LSTM, and GRU all using the same default hyperparameters. After training, I generated several outputs using different prime strings such as `Breaking`, `Area Man`, `When a`, and `This just in:` that I thought resembled headlines found in *The Onion* to see what variations I get.

Here are some example outputs I generated using the `Area Man` prime string:

RNN Output:

```
Area Man say strand in 'jerry new end food election speaking undetters solo vernoz
brief secretary demonsged
```

LSTM Output:

```
Area Man buys new his president is form clinton bad at man thinking with importance
conversation of the chara
```

GRU Output:

```
Area Man teacher irare dance mannets rurious outserve nest fan part leaframes for
sick was means burnagy pant
```

3.1.2 Evaluation

I then manually evaluated 7 generated sentences from each model by creating a rubric based on three metrics: grammar and sentence structure, sarcasm/ absurdity, and headline-like accuracy. I rated each sentence on a scale from 1 to 3 (where 3 is the best).

I used prime strings that resemble standard headlines from *The Onion* when prompting each of the models:

- `Breaking`
- `Area Man`
- `When a`
- `This just in:`

As you can see, these prompts are quite short, but were chosen to see if I can replicate similar sarcastic and punchy tone of real *The Onion* headlines.

This is the scoring rubric I selected:

- **Grammar**: Did the sentence use real words? Was it coherent or gibberish?
- **Sarcasm/Absurdity**: Did it include irony, exaggeration, or satire?
- **Headline-Like Accuracy**: Did it resemble an actual headline in tone and structure?

Average scores per model:

Model	Grammar	Sarcasm	Headline	Total (out of 9)
RNN	1.29	1.14	1.57	4.00
LSTM	2.29	1.86	2.43	6.57
GRU	1.57	1.14	1.71	4.43

Based on my evaluation, the LSTM model performed the best, scoring 6.57 out of 9. Of course, this evaluation is subjective, since I was the only evaluator, so bias isn't fully accounted for. I still tried to stay as consistent as possible. The sequences that received the highest ratings were ones that reminded me of real Onion headlines through their exaggerated tone or absurd premises.

3.2 Model Tuning

3.2.1 Hyperparameters

After choosing LSTM as my architecture of choice (since it performed the best), I tuned the model's hyperparameters to improve generation quality. To make this easier, I wrote a simple tuning script called `tune_lstm.py` that performs a basic grid search over different hyperparameter combinations. I then trained the model on the following combinations of:

- **Hidden Size**: 128, 256
- **Number of Layers**: 1, 2
- **Learning Rate**: 0.01, 0.005

I decided to fix the following hyperparameters:

- **Number of Epochs**: 500
- **Chunk Length**: 200
- **Batch Size**: 100

I then evaluated the outputs for each model using the same rubric I used before.

Model Name	Grammar	Sarcasm	Headline	Total Avg
`lstm_128_1_0.005.pt`	1.00	1.00	1.00	3.00
`lstm_128_1_0.01.pt`	1.57	1.71	1.57	4.86
`lstm_128_2_0.005.pt`	1.00	1.14	1.29	3.43
`lstm_128_2_0.01.pt`	1.43	1.57	1.29	4.29
`lstm_256_1_0.005.pt`	1.43	1.71	1.83	4.97
`lstm_256_2_0.005.pt`	**1.71**	**1.57**	**1.86**	**5.14**
`lstm_256_2_0.01.pt`	1.00	1.00	1.00	3.00

The best hyperparameter combination was `lstm_256_2_0.005.pt`, receiving the highest overall average score. I decided to use this model as my final version, and then trained it with these final hyperparameters:

- **Hidden Size**: 256
- **Number of Layers**: 2
- **Learning Rate**: 0.005
- **Chunk Length**: 200
- **Batch Size**: 100
- **Epochs**: 500

3.2.2 Final Generation and Sampling

Once my final model was selected (`lstm_256_2_0.005.pt`), I manually experimented with the temperature and generation length parameters to see how they affected text quality and style. I also felt that some outputs were too long, which took away from the punchy, headline-like structure I was aiming for.

- **Temperature** controls randomness: smaller values (e.g., `0.6`) generate safer more structured outputs, whereas larger values increase creativity but also the risk of incoherence.
- **Predict Length** controls how many characters are generated.

I ran all my samples using the following command structure:

```
python generate.py models/lstm_256_2_0.005.pt \
    --prime_str "<your prompt>" \
    --temperature <value> \
    --predict_len <value>
```

I chose to show a few of the outputs using different prompts.. some political, some random, just to see what the model would come up with. As a fan of *The Onion*, I was curious to test how close the results felt to real headlines. This part wasn't a structured evaluation but more of a trial and error approach.

Example 1
`prime_str: "Area Man", temperature: 0.6, predict_len: 80`

```
Area Man funding in phone after entire experience all gym area man to see plane
can't bel
```

Seems like lower temperature produces less absurd outputs but still captures a headline-like structure.

Example 2
`prime_str: "Area Man", temperature: 0.65, predict_len: 60`

```
Area Man selform company not entire crack of real called to get drive
```

Slightly higher temperature makes it weirder and more Onion-like. I tried the shorter length to avoid trailing characters.

Example 3
`prime_str: "Area Man", temperature: 0.65, predict_len: 60`

```
Area Man senate to see he shirt of man he chance into assive plans fo
```

Odd but has some structure and a slightly satirical tone.

Example 4
`prime_str: "When a", temperature: 0.65, predict_len: 60`

```
When a face of friend job press trump 7-year-old on during 'the tim
```

The "When a" structure works well. Pairing Trump with a 7-year-old captures the absurdity well.. although, depending on how you look at it, it might not be that far off.

Example 5
`prime_str: "Trump", temperature: 0.65, predict_len: 60`

```
Trump of outside in will first hair king president of charitiams c
```

I used "Trump" to test political figures, the phrase "hair king" makes it sound especially satirical.

Example 6
`prime_str: "Biden", temperature: 0.65, predict_len: 60`

```
Biden announces profile speech phone studio excited to see end to
```

Surprisingly readable! And sounds close to a real headline, though less overtly sarcastic.

Example 7
`prime_str: "1000", temperature: 0.65, predict_len: 60`

```
1000 all named to sell single subscrom leader like crying to keep
```

I tried a numerical prompt. The tone is dramatic and exaggerated.

These were just a few outputs I thought were worth sharing because they get surprisingly close to the kind of sarcastic and absurd tone seen in *The Onion*'s headlines. While they're not always accurate or coherent, many examples reflect the intended tone well. You can tell from the outputs that some phrases, like "announces," mimic real headline structure. Sentence flow also improves in later generations, and the model often grounds the absurdity by juxtaposing unusual words in a way that adds to the sarcasm and exaggeration, which I thought was cool.

3.3 Conclusion

In this project, I explored a different application of RNN models by using them to generate sarcastic text. My goal was to understand the role of memory in modeling sarcasm. Out of the three architectures I tested, LSTM performed the best, which I believe is due to the way it handles longer-term dependencies. This helped when generating sarcasm, especially once I tuned the hyperparameters. In general, it seems like more layers and larger hidden sizes generated better outputs.

That said, my evaluation was definitely limited. It was only myself rating the outputs, so bias is a factor. In the future, I'd want to include more evaluators, build a clearer rubric, and test a larger number of generations to get more reliable results. I'd also like to add quantitative evaluation metrics like perplexity to better assess coherence, which was something my model struggled with at times.

I also considered training a classifier to distinguish between sarcastic and non-sarcastic headlines using the full dataset (The Onion for sarcasm, HuffPost for non-sarcasm), which I think could be an effective next step.

Sarcasm is tricky at first because it doesn't always look like what it really is. It's more than just the words, but about the context behind them. That's all part of the idea of context, and it's exactly what makes sarcasm hard for machines to understand. By trying different architectures and tuning my models, I wanted to see how much of that could still come through using just text. The outputs aren't perfect, but I believe they captured some of the absurdity and exaggeration that sarcasm relies on. They're starting to pick up on the patterns that make sarcasm what it is, and hopefully just enough to make you smirk.

References

[Das23] Poulami Das. 10 hyperparameters to keep an eye on for your lstm model and other tips. Medium, 2023.

[IBM21] IBM Research. Recurrent neural networks (rnn). IBM Think, 2021.

[Kar15] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. *karpathy.github.io*, 2015.

[KS24] Rahul Kumar and Divyanshu Singh. Text generation using rnn with lstm for sarcasm detection. *IRJMETS*, 6(4), 2024.

[Mai22] Suvankar Maity. Rnn vs lstm vs gru - why do we need them? LinkedIn, 2022.

[Nar21] Pratik Narayan. Rnn vs gru vs lstm — the ultimate showdown. Medium - Analytics Vidhya, 2021.

[spr17] spro. char-rnn.pytorch. GitHub, 2017.

[Tu25] Zhuowen Tu. Cogs 185 spring 2025 lecture slides, 2025. Course Slides, UC San Diego.