José Federico Gonzlez Prior A01704642
Intelligent Systems Technologies
Ph.D. Benjamin Valdez

Text Generation using LSTM´s

Introduction

In the realm of natural language processing (NLP), Long Short-Term Memory (LSTM) networks and word embeddings have emerged as powerful techniques that revolutionized various NLP tasks. LSTMs are a type of recurrent neural network (RNN) that excel at capturing long-term dependencies and sequential patterns in text data. Word embeddings, on the other hand, represent words as dense vectors in a high-dimensional space, enabling machines to understand and reason about the meaning and relationships between words.

The use of LSTMs and word embeddings has had a profound impact on a wide range of NLP applications, including machine translation, sentiment analysis, text classification, named entity recognition, and text generation. These techniques have greatly improved the performance of these tasks by enabling models to learn more meaningful and contextualized representations of text.

One key aspect of LSTMs is their ability to overcome the vanishing gradient problem, a challenge that plagued traditional RNNs. The LSTM's unique architecture, consisting of memory cells and gates, allows it to selectively remember or forget information over long sequences, making it highly effective for tasks that require capturing dependencies over time.

Word embeddings, such as Word2Vec, GloVe, and FastText, enable us to represent words as continuous vectors in a lower-dimensional space. These embeddings capture semantic and syntactic relationships between words, allowing models to leverage this contextual information for better understanding and generalization. By leveraging word embeddings, models can handle out-of-vocabulary words, infer word similarities, and even perform arithmetic operations on words (e.g., king - man + woman = queen).

Text generation is an area where LSTMs and word embeddings have made significant strides. By training LSTMs on large text corpora, models can learn the statistical patterns and relationships within the data, allowing them to generate coherent and contextually relevant text. Text generation has found applications in chatbots, language modeling, storytelling, and even generating code or poetry.

The importance of LSTMs and word embeddings lies in their ability to capture and leverage the intricate structures and semantic relationships present in human language. By enabling machines to understand and process textual information at a deeper level, these techniques have paved the way for more advanced and intelligent NLP applications. As research in this field progresses, we can expect further advancements in text generation and other NLP tasks, driving us closer to machines that can truly comprehend and communicate in human language.

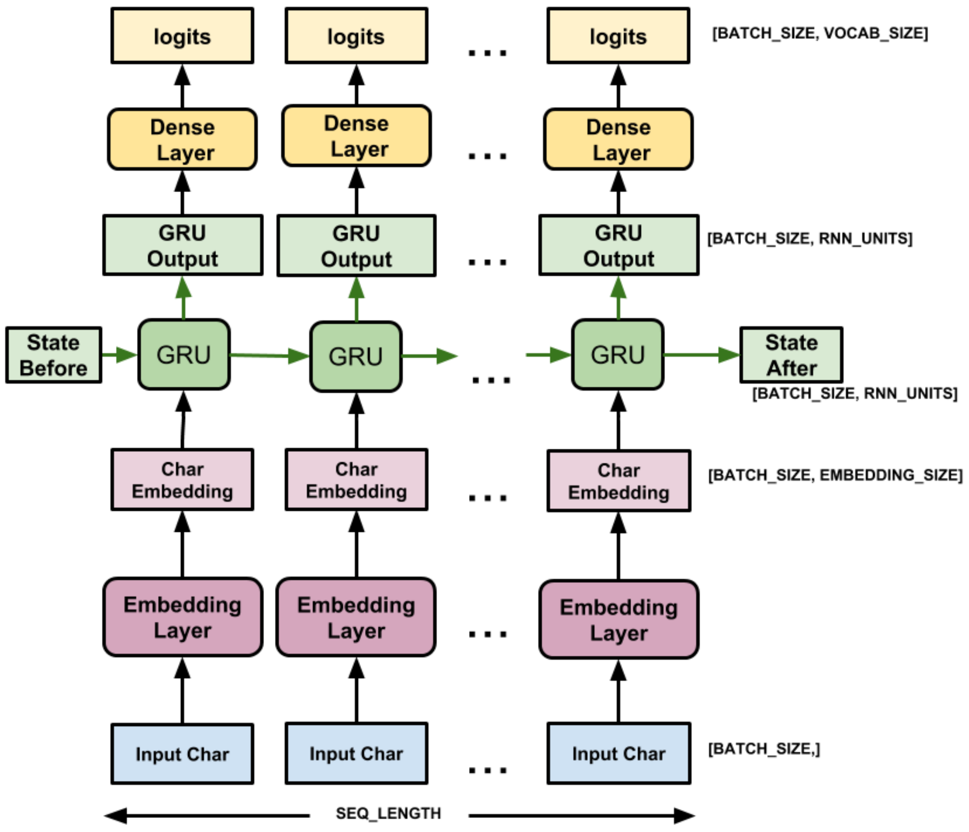


Figure 1: *Text Generation with GRU neurons*

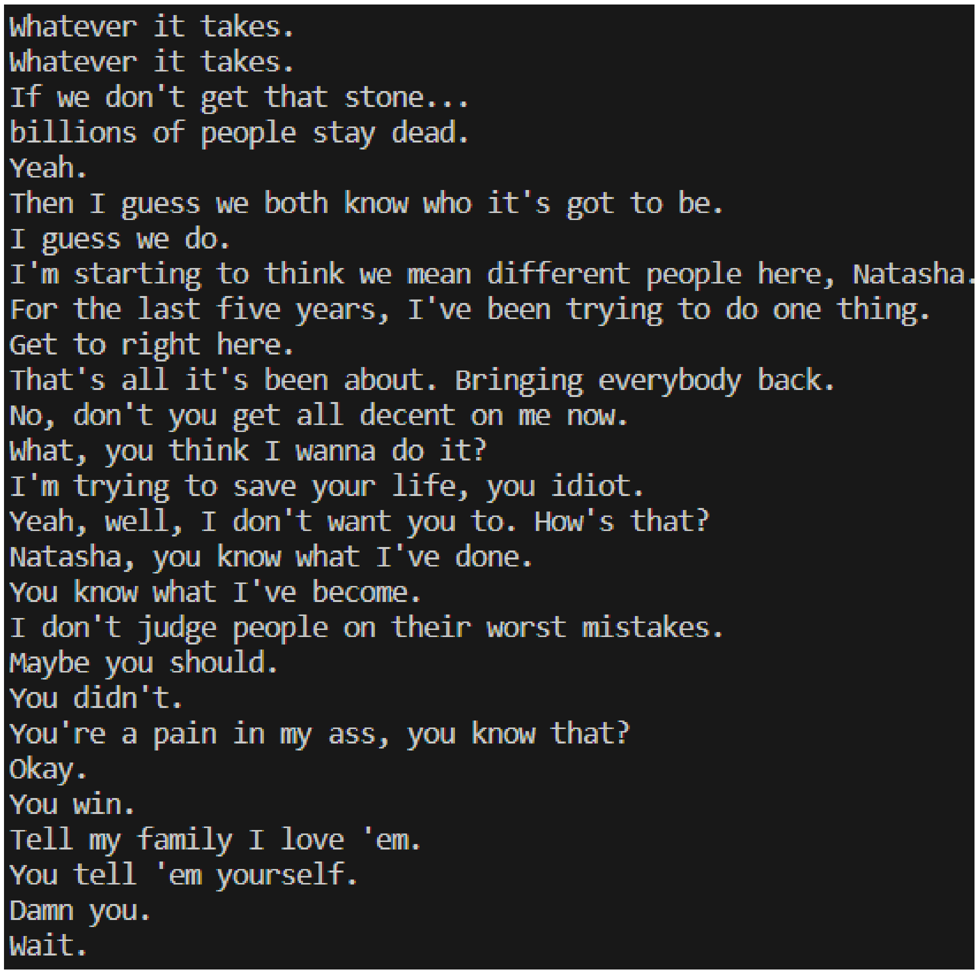
Data preparation

For this project, we chose a dataset sourced from Kaggle, which consists of the complete script of the last films in the Marvel Cinematic Universe (MCU). Among the available options, we specifically selected the script of Avengers: Endgame, considering its significance as the longest film in the MCU series.

Upon inspecting the dataset, we observed that it contains 12,828 unique words. Additionally, there are 1,862 instances where sequences are repeated. Considering these observations, we proceeded with the tokenization process. To achieve this, we employed the Spacy library followed by the Keras Tokenizer module. By tokenizing the text, we converted it into individual tokens, allowing us to represent each word numerically. Subsequently, we vectorized the words, transforming them into numerical representations suitable for inputting into our model. This vectorization process enables the model to process and learn from the textual data effectively.

With the completion of tokenization and vectorization, the data was now prepared to be fed into our model. This preprocessing step is crucial in facilitating the model's comprehension and analysis of the textual information, enabling it to capture the underlying patterns and relationships within the script of Avengers: Endgame.

By leveraging this processed data, we can now move forward with constructing and training our model. This utilization of the script data from the MCU films will provide us with the foundation to generate compelling and contextually relevant text, enhancing our understanding and exploration of the Marvel universe through automated text generation.


```
Whatever it takes.
Whatever it takes.
If we don't get that stone...
billions of people stay dead.
Yeah.
Then I guess we both know who it's got to be.
I guess we do.
I'm starting to think we mean different people here, Natasha.
For the last five years, I've been trying to do one thing.
Get to right here.
That's all it's been about. Bringing everybody back.
No, don't you get all decent on me now.
What, you think I wanna do it?
I'm trying to save your life, you idiot.
Yeah, well, I don't want you to. How's that?
Natasha, you know what I've done.
You know what I've become.
I don't judge people on their worst mistakes.
Maybe you should.
You didn't.
You're a pain in my ass, you know that?
Okay.
You win.
Tell my family I love 'em.
You tell 'em yourself.
Damn you.
Wait.
```

$Figure\ 2. Movie\ script$

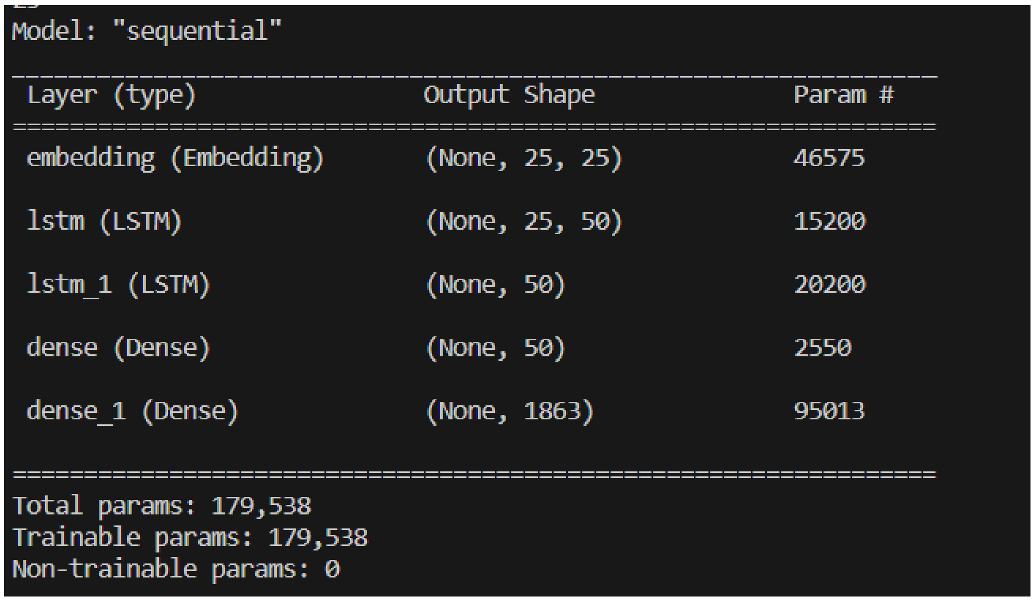
LSTM implementation

In this case, we designed a powerful sequential model to tackle the task at hand. Our model incorporates essential components, starting with an embedding layer that enables us to represent words as dense vectors, capturing their contextual meaning. This embedding layer acts as a bridge between the raw text input and the subsequent layers of the model.

José Federico Gonzlez Prior A01704642
Intelligent Systems Technologies
Ph.D. Benjamin Valdez

To capture the long-term dependencies and sequential patterns in the text data, we employed two LSTM layers, each consisting of 50 nodes. LSTMs are renowned for their ability to retain and propagate relevant information over extended sequences, making them ideal for tasks requiring temporal understanding.To further enhance the model's capacity for learning intricate relationships, we incorporated two dense layers, also comprising 50 nodes each. These dense layers enable the model to extract higher-level features from the LSTM representations, enabling more effective learning and decision-making.

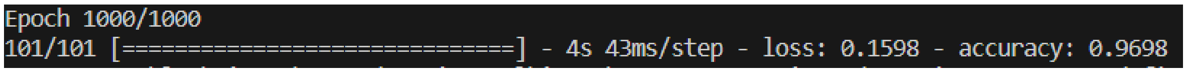
To ensure optimal training of our model, we compiled it using the categorical cross-entropy method. This loss function is well-suited for multi-class classification problems, enabling the model to effectively optimize its parameters to minimize the discrepancy between predicted and actual class labels. By combining the embedding layer, LSTM layers, dense layers, and employing the categorical cross-entropy method, our model is equipped to handle complex text data, capturing nuanced patterns and relationships. This comprehensive architecture empowers the model to generate accurate predictions and make informed decisions, ultimately advancing the field of natural language processing.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 25, 25)            46575

 lstm (LSTM)                 (None, 25, 50)            15200

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 50)                2550

 dense_1 (Dense)             (None, 1863)              95013

=================================================================
Total params: 179,538
Trainable params: 179,538
Non-trainable params: 0
```

Figure 3: LSTM model internal arquitecture

Results



```
Epoch 1000/1000
101/101 [==============================] - 4s 43ms/step - loss: 0.1598 - accuracy: 0.9698
```

Figure 4: Final accuracy and loss

The first time that we run the script we were using just 5 epochs for trial purposes, in which the accuracy of the model was about 0.02, this made the model perform terribly when we tried to enter a query because it only returned the most concurrent word, in this case was “the”. After this we decided to train the model to 100 epochs in which the accuracy was improved to 0.3, this improvement give us better text generation for short answers, for larger queries the text started to have nonsense and poor coherence.

The next thing we do was to increase the number of epochs to 500, in this case, the text generation has an accuracy of 0.7 and the improvement was drastic with the text generation, but the training time was about half an hour in a normal pc. We decided to go further and train the model for 1 thousand epochs, after a long session of nearly 2 hours we ended up with an accuracy of 0.98.

Conclusion

Finally, we can conclude thatour experiments highlighted the importance of training duration and the number of epochs in achieving better text generation results. While a higher number of epochs resulted in enhanced accuracy and improved text generation, it also increased training time. Therefore, a trade-off must be considered between training time and the desired level of accuracy and coherence in the generated text. These findings provide valuable insights for future work on text generation tasks and underline the need for careful parameter tuning and evaluation to strike the right balance between training time and performance.