NEURAL NETWORK-BASED CHARACTER-TO-SYMBOL SEQUENCE TRANSLATION FOR TEXT COMPRESSION

Nicolò Caterino,
2054849
La Sapienza
Roma
caterino.2054849@studenti.uniroma1.it

Luca Murra,
1920342
La Sapienza
Roma
murra.1920342@studenti.uniroma1.it

ABSTRACT

This project investigates the application of deep neural networks for text compression, focusing on character-to-symbol sequence translation. We propose a survey on methods based on converting human-readable text into a format compatible with existing decompression tools like ZIP, bypassing the need for intermediate representations. Several model were tested (Dense, CNN, RNN, custom) by training on a diverse dataset of text documents, including BBC articles and American high school eleventh-grade textbooks. The models are trained to produce outputs resembling the corresponding ZIP files generated by the software 7-Zip. We evaluate the models using two metrics: Levenshtein distance and a custom "unzip metric" that reflects successful decompression.

1 Introduction

In the digital age, data compression plays an increasingly crucial role, particularly in managing text data. The amount of textual information is growing exponentially, and with it the need for efficient compression methods.
While tools like ZIP software already offer a good level of compression, relying solely on traditional techniques may not be the optimal solution to fully exploit the potential of this technology.
This strategy aims to achieve higher compression efficiency while seamlessly integrating with the existing decompression infrastructure.
For our project we tried several architecture trained on the same datasets text documents. The dataset includes BBC articles from 2015-2016 and American high school eleventh-grade textbooks covering various subjects such as geography, history, and literature.
We believe this careful selection of data offers a representative sample of different writing styles and vocabulary usage, contributing to the model's generalization capabilities.
The training process involves feeding text files as input,after **"tokenization"**, to the Neural Network, guiding it to produce outputs similar to the corresponding ZIP files generated by a specific software (i.e. 7-Zip).
For evaluation, we employ two key metrics: **Levenshtein** distance to measure the similarity between the model's output and the reference ZIP file, and a custom **"unzip metric"** that reflects the successful decompression achieved by the chosen software.

2 Methodology

In this section, we discuss the methodology employed for training neural networks to compress text files from specific datasets. We utilized the LLAMA tokenizer, various architectures implemented in PyTorch Lightning, including Dense, Convolutional, RNN and our Custom. Each model was trained and evaluated to compare their performance in text compression tasks.

2.1 LLAMA: Text Tokenizer

LLAMA (Linguistic-Level Matching) is a text tokenizer designed to facilitate natural language processing tasks by breaking down input text into smaller, meaningful units called tokens. These tokens can include words, subwords, or characters, depending on the specific configuration of the tokenizer.
LLAMA is particularly useful for working with text-based information due to its ability to handle various linguistic phenomena, such as word morphology, compound words, and out-of-vocabulary terms. By breaking down text into tokens, LLAMA enables downstream models, such as neural networks, to process and understand textual data more effectively.

2.2 Dense Model

The Dense model, also known as **fully connected layers**, is a simple and straightforward architecture commonly used in deep learning tasks. Its advantage lies in its ability to learn complex patterns in data. However, for sequential data like text, Dense models may struggle to capture long-range dependencies efficiently due to their lack of inherent temporal awareness. To ensure a balanced training, we added Dropout and Batch Normalization layers.

2.3 Convolutional Model

The Convolutional Neural Network (CNN) architecture is well-suited for capturing spatial patterns in data, making it suitable for tasks like image processing. In text compression tasks, CNNs can learn hierarchical representations of the text by applying convolutional filters across different parts of the input.

2.4 RNN Model

The Recurrent Neural Network (RNN) architecture is specifically designed to handle sequential data by maintaining an internal state or memory. This makes RNNs particularly suitable for tasks where the order of input elements matters, such as text compression. In particular, we used an RNN with 2 LSTM hidden layers and a dense layer at the end.

2.5 Custom Model

The custom implementation, which combines elements from multiple architectures, was devised in an attempt to leverage the strengths of different neural network components. The architecture begins with a **convolutional layer** to extract spatial features from the input data, followed by a recurrent layer with **LSTM units** to capture temporal dependencies. Subsequently, **another convolutional layer** is employed to further refine the extracted features. **Finally, two fully connected layers with dropout and batch normalization** are added to the network for classification.

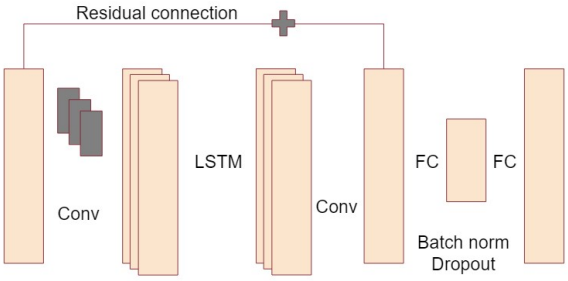

Figure 1: Architecture of custom Model

3 Dataset

In this chapter, we provide details about the datasets used for training and testing the neural network-based text-to-symbol compression model. Two distinct datasets were employed, to represent diverse real-world scenarios and ensure the model's robustness across different text genres and topics. The variation in text sizes within each dataset further challenges the model to generalize its compression capabilities across a range of input lengths. Both Dataset contain .txt files ranging from 1kb to 5kb dimension.

- **BBC Articles Dataset**, comprises approximately 2000 text samples extracted from articles published by the BBC covering various topics such as sports, business, and technology
- **School Books Dataset**, comprises approximately 3000 text samples in TXT format, with sizes ranging from 1KB to 5KB.These texts are sourced from 11th and 12th-grade school books used in the US.

4 Evaluation Metrics

To assess the performance of the neural network-based text-to-symbol compression model, two evaluation metrics are employed:

- the **Levenshtein distance**, the byte-wise difference between the strings
- the **unzip metric**, the number of outputs succesfully unzipped by 7-Zip

5 Results Analysis

In this section, we analyze the results obtained from training the neural network-based text-to-symbol compression model, with several techniques, in order to evaluate the difference between the Models.

5.1 Dense Model

The performance of the Dense model does not meet the expected standards, as depicted in Figure 2. Despite training, the loss values do not decrease significantly, with the training loss plateauing at around 600 and the validation loss around 2000. These results indicate that the model struggles to effectively learn the task of file compression and decompression.

5.1.1 Performance Metrics

The performance metrics further illustrate the performance of the Dense model:

- Total unzip metric: 0
- Total Levenshtein distance: 711,811

These metrics highlight that the model fails to successfully decompress input sequences, as evidenced by the unzip metric of 0. Additionally, the high Levenshtein distance indicates significant discrepancies between predicted and actual outputs, suggesting a lack of accuracy in the model's predictions.
Further analysis and adjustments to the model architecture and training procedure are required to improve the performance of the Dense model in file compression and decompression tasks.

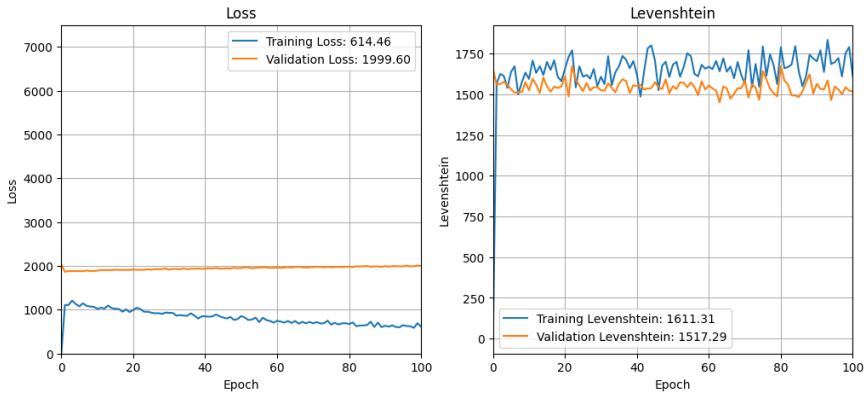

Figure 2: Training and Validation Loss for Dense Model

5.2 Convolution Model

The training and validation metrics for the Convolution model are presented in Figure 3. During the training process, we observe an unsteady decrease in the loss function, indicating that the model is trying to minimize the error between

predicted and actual outputs. However, it's noteworthy that also the Levenshtein distance decrease in an unsteady way, throughout both the training and validation phases. This observation suggests that there may be opportunities for further improvements in the model's ability to accurately predict output sequences.
After approximately 50 epochs of training, the loss function reaches a value close to 2000 for both validation and training, indicating that the model has probably converged to a local minimum in the optimization landscape.

5.2.1 Performance Metrics

The performance metrics further illustrate the performance of the Convolutional model:

- Total unzip metric: 0
- Total Levenshtein distance: 1,247,615

These metrics highlight that the model fails to successfully decompress input sequences, as evidenced by the unzip metric of 0,and also that the Levenshtein distance indicates worst performance with respect to Dense model, by almost doubling the previous result when it comes to levhenstein evaluation.

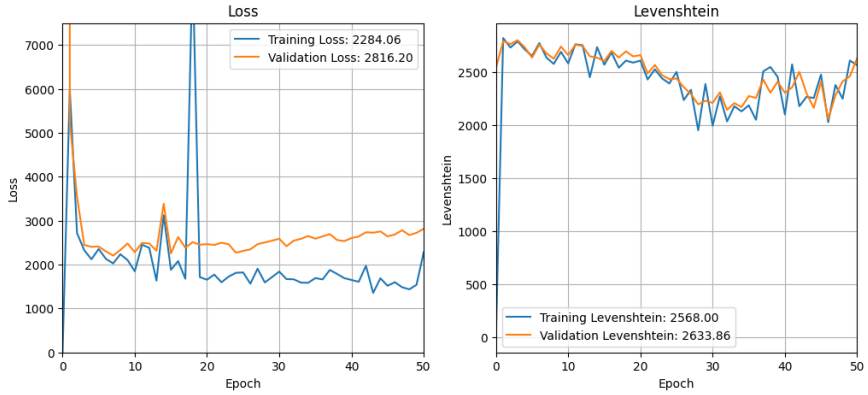

Figure 3: Training and Validation Metrics for CNN Model

Despite achieving a low loss value, the unzip metrics and Levenshtein distance metrics provide insights into the model's performance beyond the loss function alone. Specifically, the unzip metrics, which evaluate the model's ability to successfully decompress the input sequences, **remain at zero**, indicating that the model fails to unzip the files correctly. Additionally, the Levenshtein distance remains high, suggesting significant room for improvement in the model's performance.

5.3 Performance of RNN Model

The performance of the RNN model, as depicted in Figure 4, does not meet the desired expectations. Despite training for 83 epochs, the loss values show minimal improvement, suggesting difficulties in learning the underlying patterns of the data. Additionally, the high Levenshtein distance indicates significant discrepancies between predicted and actual outputs.

5.4 RNN Model Metrics

The performance metrics further highlight the challenges faced by the RNN model:

- Total unzip metric: 0
- Total Levenshtein distance: 802,429

Similar to the Dense model, the RNN model's performance is suboptimal, with both models failing to successfully decompress input sequences, as evidenced by the unzip metric of 0. The high Levenshtein distance further emphasizes the model's struggle to accurately predict output sequences.

These observations suggest that there is ample room for improvement in the RNN model's performance. Further exploration and adjustments to the model architecture, hyperparameters, or training procedure may be necessary to enhance its effectiveness in file compression and decompression tasks.

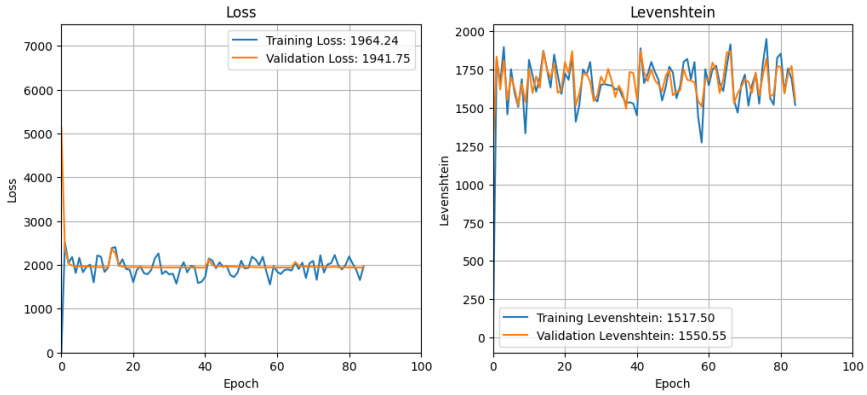

Figure 4: Training and Validation Loss for RNN Model

5.5 Custom Implementation

Despite the comprehensive design and 50 epochs of training, the performance of the custom implementation falls short of expectations. As illustrated in Figure 5, the loss values exhibit limited improvement over the course of training. Moreover, the Levenshtein distance metrics indicate poorer performance compared to both the RNN and Dense models.

5.5.1 Custom Implementation Metrics

The performance metrics for the custom implementation are as follows:

- Total unzip metric: 0
- Total Levenshtein distance: 1,009,915

These results underline the challenges encountered in integrating diverse architectural components into a cohesive framework. Despite efforts to capitalize on the strengths of individual components, the custom implementation fails to achieve the desired level of performance. However, the suboptimal results leave ample room for future improvements and optimizations.

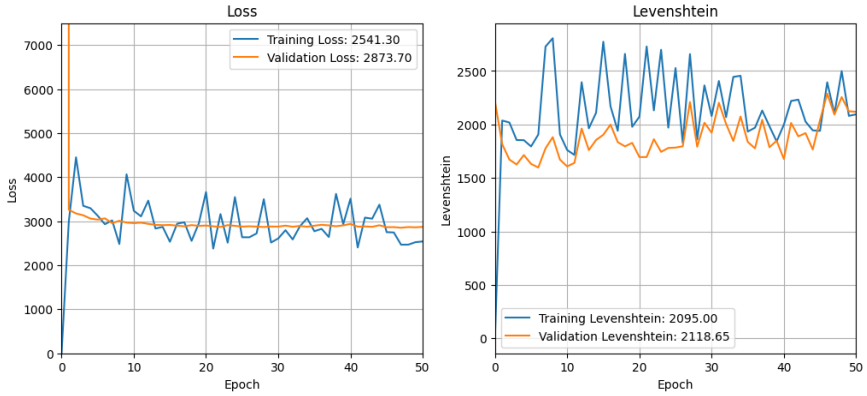

Figure 5: Training and Validation Metrics for Custom Implementation

Model	Lev. distance	Mean distance of a sample	Unzip
Dense	711,811	949.1	0
CNN	1,247,615	1,663.5	0
RNN	802,429	1,069.9	0
Ours	1,009,915	1,346.6	0

Table 1: Overall results