Reproducing Results- Predicting next Word using RNN and LSTM cells: Statistical Language Modeling

Nitul Singha[1], **Gordon Smith**[2], **Abhishek Dagur**[3]

[1]University Of West Florida

ns122@students.uwf.edu, gs68@students.uwf.edu, asd32@students.uwf.edu

Abstract. *We present a novel generative language model for structured document retrieval, which is a key task in NLP. Our model treats a document as a tree of nodes, each corresponding to a document part (e.g., title, paragraph, section). We assign a language model to each node, which is either learned from the text (for leaf nodes) or interpolated from the child nodes (for inner nodes). This way, we capture the structure and context of the document. We use our model to rank documents and their parts based on structural queries, which specify the desired document part and its content. Our model offers a flexible and effective way to retrieve documents based on their structured content.*

1. Project Contributions

Abhishek Dugar *(Research Project Manager)*

Provided leadership for the group to stay on track. Wrote multiple parts of the report and wrote about an analysis and impact of the paper. Helped run the models with different parameters to produce better results.

Gordon Smith *(Data Analyst and Documentation)*

Found datasets that could be used to train the models. Created code for the many-to-one and many-to-many models and their predictions. Added figures and descriptions about code to the report.

Nitul Singha *(Principal Investigator)*

Wrote multiple parts of the report including the definitions and methodology. Helped run the models with different parameters to produce better results. Created final report format and added diagrams of the models.

2. Introduction

Language modeling (LM) is a fundamental task in natural language processing (NLP) that involves assigning likelihood distributions to sequences of words. The goal is to capture the contextual dependencies between words within a sequence, enabling the model to distinguish between similar phrases that convey different meanings. For instance, phrases like "recognize speech"and "wreck a pleasant beach"may sound alike but have distinct interpretations. LM not only encapsulates the complexities of language, such as grammatical structures, but also encapsulates significant amounts of information contained within corpora.

In recent years, deep learning techniques, particularly Recurrent Neural Networks (RNNs), have revolutionized LM research. Unlike simpler models like N-grams, which

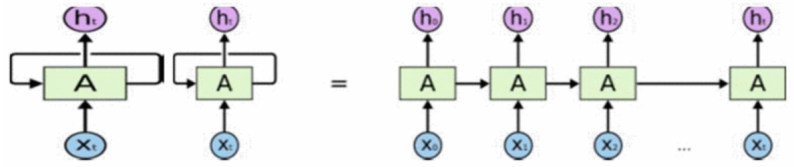
rely on limited historical information to predict the next word, RNNs excel at capturing long-range dependencies in sequences. They achieve this by employing loops within the network architecture, allowing information to persist across time steps. This recurrent nature enables RNNs to effectively model sequential data, making them indispensable for various NLP tasks.

Conceptually, RNNs can be visualized as multiple copies of standard neural networks, each passing messages to the next in a chain-like fashion. Unrolling these loops reveals their inherent association with sequences and lists, making them well-suited for tasks involving sequential data processing.

One of the key advantages of RNNs is their ability to remember past inputs through internal memory, which is crucial for tasks requiring sequential information processing. This memory mechanism enables RNNs to retain context over time steps, facilitating tasks such as language generation and sequence prediction.

Mathematically, RNNs define a recurrence relation over time steps, where the current state is computed based on the current input and the previous state. This recursive formulation, depicted by the formula provided, allows RNNs to capture temporal dependencies and make predictions over sequential data.

RNNs have been instrumental in powering various applications, including virtual assistants like Apple's Siri and Google Voice Search. Their ability to remember past inputs and leverage sequential information has made them indispensable for tasks involving natural language understanding and generation.



In summary, language modeling with RNNs is a critical aspect of NLP research, enabling the modeling of contextual dependencies in sequences of words. RNNs excel at capturing long-range dependencies and retaining information over time steps, making them essential for a wide range of applications in language processing and beyond.

3. Definitions of Some Terms

Here are definitions of Keywords used in the paper.

Epoch: within the neural network terminology: one epoch = one forward and one backward pass of all the training examples.

Iterations: Iteration could be a term utilized in machine learning and indicates the amount of times the algorithm's parameters are updated. Number of passes, every pass by means of [batch size] number of examples. One pass = one forward pass + one backward pass (we don't count the forward pass and backward pass as 2 completely separate passes).

Batch size: the amount of training examples in one forward/backward pass. The larger the batch size, the more the memory area you will need.

LSTM: Long short-term memory (LSTM) is artificial recurrent neural network

(RNN) architecture. Unlike regular feed-forward neural networks, LSTM has feedback connections that make it a "general purpose computer" (that is, it can compute anything that a Turing machine can). It cannot only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as connected handwriting recognition or speech recognition.

LSTM Memory: LSTM memory cell refers to one of the two halves of an LSTM layer's hidden state: It's the portion of the hidden state that is only modified by addition/-subtraction and scaling, and so tends to preserve information for a relatively long time. In this reading the cell is a variable (not a function) which takes on different values at each time step.

Dropout: Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. Dropout is easily implemented by randomly selecting nodes to be dropped-out with a given probability (e.g. 20%) each weight update cycle. Dropout is only used during the training of a model and is not used when evaluating the skill of the model.

Initialization Scale: It is the initial weight of all neurons using heuristic algorithms.

4. Literature Review

J. Goodman et al. [4], S.F. Chen et al. [2], and J.T. Goodman et al. [4], as well as R. Kneser [5] and H. Ney [5], have discussed various language modeling (LM) approaches, including the N-gram model and neural language models (NLMs) such as the feed-forward neural network based LM and the recurrent neural network based LM. The N-gram model, while based on the Markov assumption, faces sparsity issues when dealing with unseen combinations in the training corpus. This challenge has led to the development of smoothing techniques, although no perfect solution exists. On the other hand, NLMs, also known as continuous space LM, aim to address sparsity and context limitations. Recurrent neural network based LM, in particular, has shown promising performance due to its ability to capture long-range dependencies. However, both approaches have their limitations. The N-gram model suffers from the curse of dimensionality, while the assumption of shared autonomy of sentences in a corpus may not hold true in larger context LM. Nonetheless, larger context LM has shown improvements in reducing confusion and enhancing sentence understanding compared to LM without context information [3].

5. Methodology

Neural networks have been extensively utilized for predicting current or subsequent words, a crucial aspect of natural language processing (NLP) [1]. Language modeling stands out as a vital task within NLP. Initially, an environment is established to mimic typing behavior, capturing the keystrokes of users. Subsequently, deep-layered convolutional neural networks are employed to model language patterns. The incorporation of a character-to-word model proves beneficial in this context. Currently, the focus is on predicting the next word based on the preceding 40 characters. For autocorrect functionalities, recurrent neural networks (RNNs) are adept at modeling grammatical and spelling errors using

training data. At its core, an RNN is essentially a densely connected neural network [6]. The fundamental distinction from traditional feedforward networks lies in the inclusion of temporal aspects; specifically, the output of the hidden layer in an RNN is fed back into itself. Visual aids, such as diagrams, can aid in comprehension.

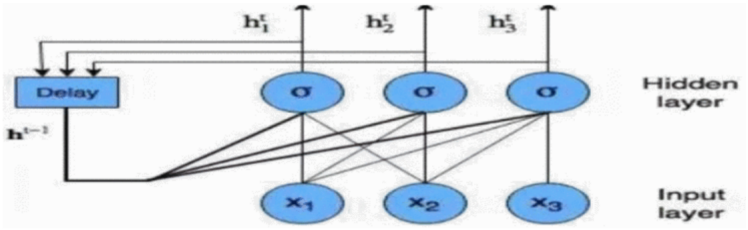


Figure 3

In Figure 3, a direct recurrent neural network is depicted, featuring three input nodes. These input nodes feed into a hidden layer, where sigmoid activations are applied, similar to a conventional densely connected neural network. What follows is particularly intriguing: the output of the hidden layer is then fed back into the same hidden layer. Through an abstract delay block, the output of the hidden layer at time step t-1 is introduced into the hidden layer. This mechanism allows for the modeling of temporal or sequence-dependent information.

A notable application of this architecture is in predicting text sequences. Consider the following text sequence: "A woman walked into a bar, and she said 'Can I have a drink please?'. The bartender said 'Certainly '."The placeholder can be filled with various possibilities, such as "miss"or "ma'am,"among others. Additionally, terms like "sir"or "Mister"could also be appropriate alternatives. To correctly determine the gender of the noun, the neural network needs to "remember"that the preceding two words, indicating the possible gender (e.g., "woman"and "she"), were used. This flow of information through time (or sequence) in a recurrent neural network is illustrated in the figure below, depicting the sequence unfolding.

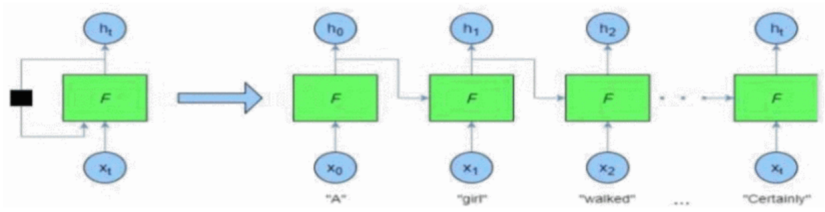


Figure 4

On the left-hand side of the provided figure (Fig. 4), a similar depiction as the main one is shown, illustrating all the nodes explicitly. What the previous figure fails to explicitly indicate is that we provide limited-length sequences to such networks - allowing us to unroll the system as depicted on the right-hand side of the given figure (Fig. 4). This unrolled system allows for a large series of data to be fed into the recurrent neural network [8].

For instance, initially, we feed the word vector for "A"(and similarly for subsequent word vectors) to the network F - the outputs of the nodes in F are fed into the "next"network and also serve as the initial output (h0). Subsequently, the network (though

it's essentially the same network) F at time t=l takes the subsequent word vector for "women"and the previous output h0 into its hidden nodes, producing the subsequent output h1, and so forth.

Now, turning back to recurrent neural networks themselves. Recurrent neural networks are highly versatile. In the implementation shown in Fig. 4, a many-to-many model is employed - in other words, we have the input sequence "A woman walked into a bar..."and a multitude of outputs - h0 to ht. An alternative option is one-to-many, where one input, such as "women,"is provided, and multiple outputs h0 to ht are predicted (for example, attempting to generate sentences based on a single starting word). Another configuration is many-to-one, where several words are provided as input, such as the sentence "A woman walked into a bar, and she said 'Can I have a drink please?'. The bartender said 'Certainly'", and the prediction is made for the subsequent word, i.e., .

The figures below, Fig. 5 and Fig. 6, illustrate examples of one-to-many and many-to-one configurations, respectively (the words adjacent to the outputs are the target words that we may provide during training).

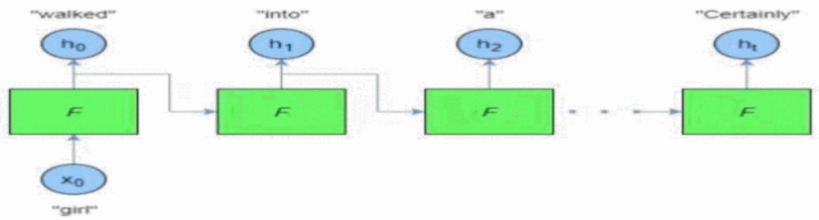


Fig. 5: One-to-many configuration of recurrent neural network.

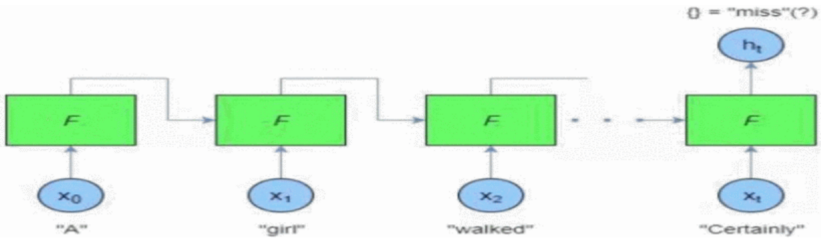


Fig 6: Many-to-one configuration of recurrent neural network.

6. Code and Data

The code for this project is split into two different files. One file creates a many-to-one $LSTM$ model to predict the next character and the other part creates a many-to-many $LSTM$ model to predict the next sequence of characters. The many-to-one model is trained on text from the book Beyond Good and Evil by $Friedrich$ Nietzsche. Sequences of 40 characters were taken from the book and used as the input sequences for the model. The next character after the sequence of 40 characters were all taken from the book and used as the target variable for the model. The many-to-many model is trained on the text from the book Alice in Wonderland by Lewis Carroll. Sequences of 40 characters were taken from the book and used as the input sequences for the model. The next 40 characters

after the sequence of 40 characters were all taken from the book and used as the target variable for the model

The Alice in Wonderland dataset can be found at: https://www.kaggle.com/datasets/chandan2495/alice-in-wonderland-gutenbergproject

Nietzsche's bibliography dataset is available at: https://www.kaggle.com/datasets/akouaorsot/nietzsches-bibliography

Source Code at :https://github.com/NSingha07/IDC6146-Group12

7. Summary of Findings

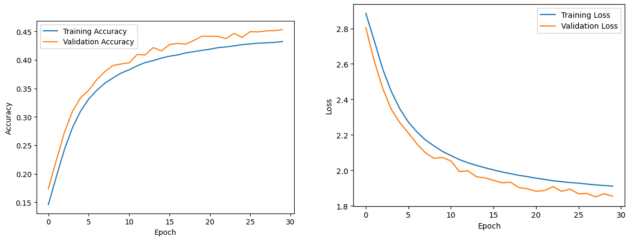
7.1. Many-to-One model

The presented work includes the implementation and analysis of two distinct language models: a many-to-one model and a many-to-many model. The many-to-one model architecture is showcased, utilizing an LSTM layer configured with $return_sequence = True$ to generate output solely for the last timestep, thus defining it as a many-to-one model. Additionally, dropout layers are incorporated to mitigate overfitting, while the final dense layer utilizes the softmax activation function to yield probabilities for individual characters.

```
# Create the model
model = Sequential()
model.add(LSTM(num_neurons, return_sequences=False, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(len(unique_characters), activation='softmax'))
```

```
Epoch 28/30

1868/1868 [==============================] - 173s 92ms/step - loss: 1.9181 - accuracy: 0.4300 - val_loss: 1.85
05 - val_accuracy: 0.4511

Epoch 29/30

1868/1868 [==============================] - 171s 92ms/step - loss: 1.9147 - accuracy: 0.4309 - val_loss: 1.86
82 - val_accuracy: 0.4515

Epoch 30/30

1868/1868 [==============================] - 172s 92ms/step - loss: 1.9112 - accuracy: 0.4325 - val_loss: 1.85
46 - val_accuracy: 0.4531
```

The training process of the many-to-one model is depicted, culminating in a final training accuracy of 0.4325 and testing accuracy of 0.4531, with corresponding losses of 1.9112 and 1.85, respectively. Plots illustrating the progression of training and validation accuracies, as well as losses over 30 epochs, indicate an initial rapid improvement followed by a more gradual convergence.




```
Original Sequence:  endeavoured go bottom question pessimis

Predicted Sequence: s snaer in
```

Predictions from the many-to-one model are showcased, albeit with limited accuracy, demonstrating successful identification of certain words such as "in."

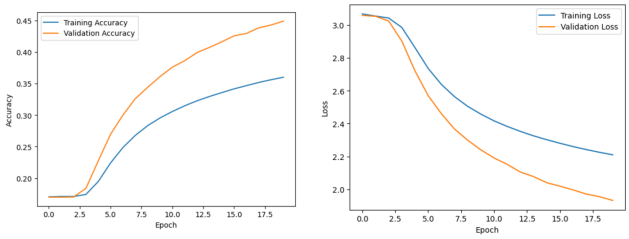
7.2. Many-to-Many model

The many-to-many model, as presented, comprises $LSTM$ layers configured with $return_sequence = True$ to generate full output sequences for each input, making it a many-to-many model. Similar to the many-to-one model, dropout layers are included for regularization, and the final dense layer employs $softmax$ activation to produce character probabilities.

```
# Create the model
model = Sequential()
model.add(LSTM(num_neurons, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(num_neurons, return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(num_distinct_chars, activation='softmax'))
```

```
Epoch 18/20

1216/1216 [==============================] - 611s 502ms/step - loss: 2.2413 - accuracy: 0.3516 - val_loss: 1.9
703 - val_accuracy: 0.4383

Epoch 19/20

1216/1216 [==============================] - 607s 499ms/step - loss: 2.2246 - accuracy: 0.3560 - val_loss: 1.9
542 - val_accuracy: 0.4427

Epoch 20/20

1216/1216 [==============================] - 592s 487ms/step - loss: 2.2094 - accuracy: 0.3600 - val_loss: 1.9
315 - val_accuracy: 0.4489
```

Training results for the many-to-many model reveal a final training accuracy of 0.36 and testing accuracy of 0.4489, with respective losses of 2.2094 and 1.9. Graphical representations of training and validation $accuracies$, as well as losses over 20 epochs, indicate ongoing improvements, suggesting potential further enhancements with additional epochs.




```
Input Sequence: eally, my dear,

you must cross-examine t

Predicted Characters:       w  iteed  rit quute aikes me for t
```

Predictions from the many-to-many model exhibit similar limitations in accuracy, yet demonstrate the model's capability to generate sequences, albeit with room for refinement.

8. Project Enhancement

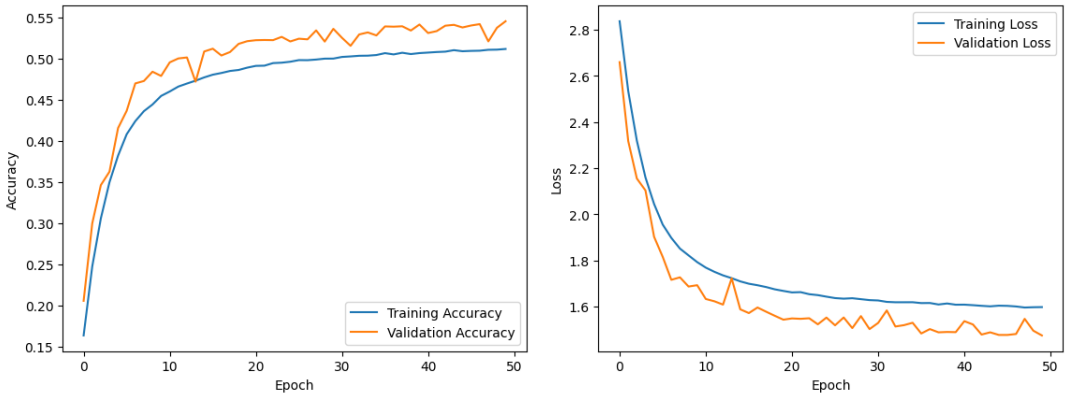
8.1. Many to One

Based on our analysis, we propose several avenues for enhancing the project. Firstly, further $hyperparameter$ tuning is recommended to optimize model performance. This

entails fine-tuning parameters such as learning rate, batch size, and optimizer selection to achieve better convergence and generalization.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 40, 128)           66560

 dropout_1 (Dropout)         (None, 40, 128)           0

 lstm_2 (LSTM)               (None, 40, 128)           131584

 dropout_2 (Dropout)         (None, 40, 128)           0

 lstm_3 (LSTM)               (None, 128)               131584

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 38)                4902

=================================================================
Total params: 334630 (1.28 MB)
Trainable params: 334630 (1.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Additionally, the implementation of regularization techniques, such as early stopping and dropout regularization, is crucial to mitigate $overfitting$ and enhance the robustness of the models.



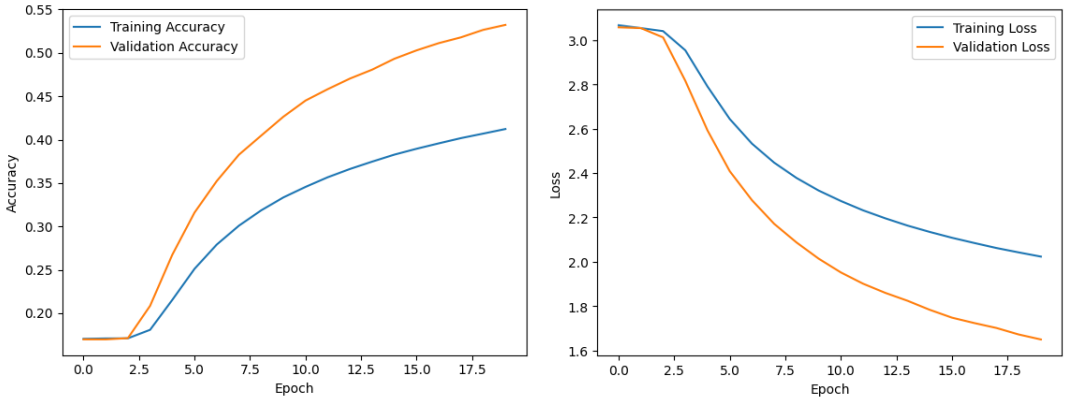
Furthermore, exploring alternative $LSTM$ architectures, including bidirectional $LSTMs$ and attention mechanisms, could provide valuable insights into improving model effectiveness and capturing more intricate patterns in the data. Employing ensemble methods, such as averaging or stacking predictions from multiple models, offers a promising approach to improve overall performance by leveraging diverse model predictions. Lastly, incorporating data augmentation techniques to increase the diversity of training examples can enhance model generalization and improve performance on unseen data. These recommendations collectively provide a comprehensive strategy for advancing the project and achieving superior results in sequential data prediction tasks.

8.2. Many to Many

We employ a new sequential $LSTM$ model with 5 layers with 20% dropout rate. The model has 1330493 total parameters, with all of them being trainable. The model was trained for 20 epochs, with each epoch taking approximately 759 seconds (12 minutes and 39 seconds) to complete.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 40, 256)           264192

 dropout_2 (Dropout)         (None, 40, 256)           0

 lstm_3 (LSTM)               (None, 40, 256)           525312

 dropout_3 (Dropout)         (None, 40, 256)           0

 lstm_4 (LSTM)               (None, 40, 256)           525312

 dropout_4 (Dropout)         (None, 40, 256)           0

 dense_1 (Dense)             (None, 40, 61)            15677

=================================================================
Total params: 1330493 (5.08 MB)
Trainable params: 1330493 (5.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The training loss decreased from 2.1406 to 2.0230, indicating a slight improvement in the model's performance. The training accuracy increased from 0.3778 to 0.4122, indicating a small improvement in the model's ability to predict the correct characters.



The validation loss decreased from 1.8348 to 1.6491, indicating a slight improvement in the model's performance on the validation dataset. The validation accuracy increased from 0.4753 to 0.5323, indicating a small improvement in the model's ability to predict the correct characters on the validation dataset.

9. Conclusion

In this project, we explored and analyzed two distinct sequential language models: a many-to-one model and a many-to-many model. Through comprehensive evaluations of their architectures, training processes, and predictive capabilities, we gained insights into their strengths, limitations, and potential avenues for enhancement.

The many-to-one model demonstrated moderate success in generating sequences, achieving training and testing accuracies of 0.4325 and 0.4531, respectively. Despite its ability to capture certain patterns in the data, the model exhibited limitations in accurately predicting sequences, suggesting room for improvement.

Similarly, the many-to-many model showcased promising capabilities in sequence generation, with training and testing accuracies of 0.36 and 0.4489, respectively. While the model exhibited improvements over the many-to-one architecture, it also faced challenges in accurately predicting sequences, indicating opportunities for refinement.

To enhance both models, we proposed several strategies, including hyperparameter tuning, regularization techniques, exploration of alternative LSTM architectures, ensemble methods, and data augmentation. These strategies aim to improve model convergence, generalization, and predictive performance, ultimately advancing the project's effectiveness in sequential data prediction tasks.

Furthermore, we implemented enhancements to the many-to-many model, incorporating a deeper LSTM architecture with dropout regularization. While these enhancements resulted in modest improvements in training and validation metrics, they underscored the potential for iterative refinement and optimization in sequential modeling tasks.

In conclusion, this project provides a foundational understanding of sequential language modeling techniques, along with actionable insights and recommendations for further improving model performance. By leveraging these insights and employing iterative refinement strategies, we can continue to advance the project and achieve superior results in sequential data prediction applications.

Referências

[1] James Allen. *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc., 1995.

[2] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[3] Jason Brownlee. Gentle introduction to statistical language modeling and neural language models - machinelearningmastery.com. https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/, 2024. Accessed: 2024-02-16.

[4] Joshua T. Goodman. A bit of progress in language modeling. *Computer Speech amp; Language*, 15(4):403–434, October 2001.

[5] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *1995 international conference on acoustics, speech, and signal processing*, volume 1, pages 181–184. IEEE, 1995.

[6] Fu-Lian Yin, Xing-Yi Pan, Xiao-Wei Liu, and Hui-Xin Liu. Deep neural network language model research and application overview. In *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 55–60. IEEE, 2015.