Implementing an Recurrent Neural Network to Generate Words

Atieh Armin[1]

[1] *Drexel University, aa4762@drexel.edu*

March 19, 2024

1 ABSTRACT

This paper explores the implementation of a character-level Recurrent Neural Network (RNN) for the task of word generation, demonstrating the model's potential in language processing applications. Utilizing the English Word Frequency Dataset from Kaggle [1], we designed a basic one-layer RNN architecture, focusing on optimizing its hyperparameters to enhance performance. Through experimentation, we determined optimal settings for learning rate, hidden node count, sequence length, and loss threshold. Our results indicate that, despite its simplicity, the RNN model effectively generates coherent English word sequences. Our findings contribute to the broader dialogue on neural networks' capability in language modeling, highlighting both achievements and areas for future investigation.

2 BACKGROUND

2.1 History of Language Models

In today's fast-changing tech world, the idea of Large Language Models (LLMs) has become really exciting for people who love technology and even those who just use it daily. These amazing pieces of artificial intelligence can understand human language and create text that looks like someone wrote it. As we dive deeper into the world of AI, it's essential to know about the basic ideas and the significant advances that have brought us to where we are now.

The story starts in the 1990s and early 2000s with N-gram models [2]. These models were simple but valuable. They guessed the next word in a sentence by looking at the words before it. This method was excellent for understanding the language context and what words mean together. N-gram models showed us how important it is to look at the context of words, setting the stage for more complex ways to understand language.

Even though N-gram models were great for voice recognition because they were simple and fast, they had problems. They weren't good at guessing new combinations of words they hadn't seen before or understanding long sentences [6]. Neural Networks [1, 4, 5] fixed these problems by learning word meanings more flexibly, leading to better performance.

In the 1990s, another type of network called Convolutional Neural Networks (CNNs) started being used for pictures, but they were also used for some language tasks, like sorting texts. Finally, Recurrent Neural Networks (RNNs) were proposed, which was a big step forward because they could follow the flow of language [7].

Unlike feedforward networks, which only have history represented by the context of N-1 words, recurrent networks can remember much more. This means they could use past information to understand what comes next in a sentence. This memory ability made recurrent networks better at dealing with different parts of a sentence [4].

All these developments, including the very first ideas like Perceptron in 1960 and later RNN, LSTM, and CNN, have shaped how we understand and work with human language today.

2.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data, such as time series or natural language text. They achieve this by using a hidden state updated for each time step of the input sequence, allowing the network to maintain a memory of previous inputs. This paper will focus on basic one-layer character-level RNNs [3].

These networks consist of an input layer, a hidden layer, and an output layer. The input layer takes in a one-hot encoded vector representing a character in the input sequence. This vector is multiplied by a weight matrix W_{ax} to produce a hidden state vector a. The hidden state vector is then passed through a non-linear activation function and updated for each time step of the input sequence. The updated hidden state is then multiplied by a weight matrix W_{ya} to produce the output probability distribution over the next character in the sequence.

Now, we will look more closely at the forward and backward propagation. During forward propagation, the input sequence is processed through the RNN to generate an output sequence. At each time step, the hidden state and the output are computed using the input, the previous hidden state, and the RNN's parameters.

At time step t, the input to the RNN is x_t, and the hidden state at time step $t-1$ is a_{t-1}. So, the hidden state at time step t is computed as:

$$a_t = tanh(W_{aa}a_{t-1} + W_{ax}x_t + b_a) \tag{1}$$

Where W_{aa} is the weight matrix for the hidden state, W_{ax} is the weight matrix for the input, and b_a is the bias vector for the hidden state. Also, the output at time step t is computed as:

[1] https://www.kaggle.com/datasets/rtatman/english-word-frequency

$$y_t = Softmax(W_{ya}a_t + b_y) \tag{2}$$

Where W_{ya} is the weight matrix for the output, and b_y is the bias vector for the output.

The objective of training an RNN is to minimize the loss between the predicted and ground truth sequences. Backward propagation calculates the gradients of the loss concerning the RNN's parameters, which are then used to update the parameters using an optimization algorithm such as Adagrad or Adam.

At time step t, the loss with respect to the output y_t is given by:

$$\frac{\partial L}{\partial y_t} = -\frac{1}{y_{t,i}}, if i = t_i, else 0 \tag{3}$$

where L is the loss function, $y_{t,i}$ is the ith element of the output at time step t, and t_i is the index of the true label at time step t. The loss concerning the hidden state at time step t is calculated by:

$$\frac{\partial L}{\partial a_t} = \frac{\partial L}{\partial y_t} W_{ya} + \frac{\partial L}{\partial h_{t+1}} W_{aa} \tag{4}$$

Where $\frac{\partial L}{\partial a_t}$ is the gradient of the loss with respect to the hidden state at the next time step, which is backpropagated through time.

The gradients concerning the parameters are then computed using the chain rule:

$$\frac{\partial L}{\partial W_{ya}} = \sum_t \frac{\partial L}{\partial y_t} a_t \tag{5}$$

$$\frac{\partial L}{\partial b_y} = \sum_t \frac{\partial L}{\partial y_t} \tag{6}$$

$$\frac{\partial L}{\partial W_{ax}} = \sum_t \frac{\partial L}{\partial a_t} \frac{\partial a_t}{\partial W_{ax}} \tag{7}$$

$$\frac{\partial L}{\partial W_{aa}} = \sum_t \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_{aa}} \tag{8}$$

$$\frac{\partial L}{\partial b_a} = \sum_t \frac{\partial L}{\partial a_t} \frac{\partial h_t}{\partial b_a} \tag{9}$$

$$\frac{\partial a_t}{\partial W_{ax}} = x_t \tag{10}$$

$$\frac{\partial a_t}{\partial W_{aa}} = a_{t-1} \tag{11}$$

$$\frac{\partial a_t}{\partial b_a} = 1 \tag{12}$$

These gradients are then used to update the parameters of the RNN utilizing an optimization algorithm such as gradient descent, Adagrad, or Adam.

3 METHOD

3.1 Architecture

The chosen architecture for this study is a basic one-layer RNN consisting of an input layer, a hidden layer, and an output layer. This structure supports the sequential nature of language by updating its hidden state for each time step, thereby maintaining a form of memory. We used Xavier Initialization to initialize our weights and used the L2 regularization technique to improve the model's performance and generalization capability. We searched several options for each Hyperparameter to optimize the model's performance. To keep our options limited, we relied on the decisions of prior works [2].

Finally, we got the best result with a learning rate 1e-3, 100 hidden nodes for complexity and manageability, a sequence length of 25 to balance computational efficiency with context capture, and a loss threshold of 3.46 as a convergence criterion. The Adagrad algorithm was employed for weight updates, enhancing the model's ability to adapt over time.

[2]https://gist.github.com/karpathy/d4dee566867f8291f086

Table 1: Evaluation Process

Iteration	Generated text	Loss
500	S, Our, Frr, Th, Ill	52.631950
1000	Ca, Nd, Will, Nd, Hos	33.460868
1500	Has, Page, You, S, This	21.729475
2000	Th, Reh, Ndw, Home, Lll	14.589912
2500	Our, Seht, Our, But, S	10.240379
3000	Lll, Res, Our, But, More	7.582041
3500	Thit, Ve, Ndw, Cae, Ndw	5.965759
...	...	...
11500	Thrthte, More, Hls, Seur, Ndw	3.469066
12000	Cat, Ree, Hes, Ndw, Th	3.466681
12500	But, But, Thes, But, Lle	3.467715
13000	Ndw, Res, All, Thit, Hom	3.467447

3.2 Evaluation

The model underwent rigorous evaluation utilizing the English Word Frequency Dataset from Kaggle [3], derived from the Google Web Trillion Word Corpus. The dataset, which contains over 333K of the most commonly used single words in the English language, provided a robust foundation for assessing the model's predictive capabilities. Key performance indicators were the loss measurement and the model's ability to generate coherent word sequences, highlighting the influence of selected hyperparameters on learning efficacy.

To evaluate our model, we print progress every 500 iterations by generating a text sample and evaluating the loss. Table 1 shows the model's performance through these iterations. With more hidden nodes, our model got more complicated, and the loss would never be lower than 50. Moreover, with a higher learning rate, we got exploding gradients, and the loss became infinity.

However, with the final setting, the loss is decreasing, as shown in table 1. After 13000 iterations, the training process is terminated since the loss is lower than our threshold. It is important to note that the loss won't get lower, and we decided on the threshold to be 3.46 after testing the model several times.

4 ANALYSIS

We implemented a prediction method to predict a few words to be able to analyze the results of our model. As shown in Figure 1, we give the model the start sequence, and it will predict the rest of the word. The results are promising as it can produce sequences similar and close to the real world.

Overall, implementing a basic one-layer RNN for word generation yielded promising results, showcasing the model's potential to generate plausible English words. Comparative analysis with traditional and neural language models underscored the RNN's superior capability in capturing sequential dependencies and generating text. However, challenges such as the vanishing and exploding gradient problems and difficulty handling long-term dependencies were identified, marking areas for further research.

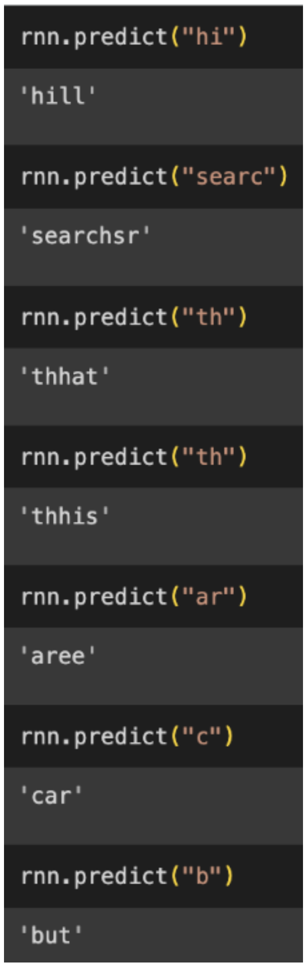
5 FUTURE WORK

Future work will explore advanced models such as Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformers to overcome the identified challenges. These models offer mechanisms to address the limitations of basic RNNs, potentially enhancing the model's ability to handle longer sequences and improving overall performance.

6 CONCLUSION

In this study, we looked into how a simple Recurrent Neural Network (RNN) can generate words by learning from sequences of characters. Our work showed that even a basic RNN model could produce text that makes sense, highlighting the power of RNNs in working with language. We tested the model using a dataset of common English words and adjusted its settings to see how well it could learn to predict new words. Choosing the correct settings, or hyperparameters, was vital to making the model work well. However, during this project, we also noticed some challenges. The model sometimes struggled with learning from long pieces of text and had issues with gradients, which are part of how it learns. Newer models like LSTM networks, GRUs, and Transformers could help solve these issues, making RNNs even better at handling language.

[3]https://www.kaggle.com/datasets/rtatman/english-word-frequency



```
rnn.predict("hi")
'hill'
rnn.predict("searc")
'searchsr'
rnn.predict("th")
'thhat'
rnn.predict("th")
'thhis'
rnn.predict("ar")
'aree'
rnn.predict("c")
'car'
rnn.predict("b")
'but'
```

Figure 1: Results

REFERENCES

[1] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A Neural Probabilistic Language Model. In *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp (Eds.), Vol. 13. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

[2] Adria Cabello. 2023. The Evolution of Language Models: A Journey Through Time. (2023). https://medium.com/@adria.cabello/the-evolution-of-language-models-a-journey-through-time-3179f72ae7eb

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[4] Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*. https://api.semanticscholar.org/CorpusID:17048224

[5] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM Neural Networks for Language Modeling. In *Interspeech*. https://api.semanticscholar.org/CorpusID:18939716

[6] Liu Z Pang Y Wang Y Zhai C Peng F Wang Y, Huang H. 2019. Improving N-gram language models with pre-trained deep transformer. (2019). https://arxiv.org/pdf/1911.10235.pdf

[7] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923* (2017).