Optimal Neural Networks for Representing Associative Memories

Maneeshika Madduri, Brandon Pratt, Pavithra Rajeswaran

ABSTRACT

Animals must perform particular sequences of actions in order to communicate and move. These sequences are encoded by short-term associative memory, a small string of prior actions, where prior actions gate the response to the next input. The encoding of these short-term associative memories in the nervous system is not fully understood. We aim to model this biological phenomenon using neural networks. We plan to investigate different structures of neural networks to encode the sequence of syllables in "hello world" and assess the efficiency of these networks. The neural networks tested will be a simple RNN, LSTM and GRU. The GRU performs better in terms of the number of training epochs required to produce a 100% prediction accuracy, whereas, the simple RNN and LSTM perform better than the GRU with fewer units in the recurrent layer. All models are unable to predict "Hello World" given a seed fewer than four characters. The GRU model generalizes better than the LSTM and simple RNN models in predicting a variety of target sequences that differ in their number of occurrences in the training data. The implications of this project would be to reveal potential network structures that may naturally encode short-term memory in the nervous system.

INTRODUCTION

The ABCs are always recited in the same way. A, then B, then C and all the way to Z. But, try saying the alphabet backwards or starting in the middle and it's much harder. That's due to sequence learning. Sequence learning leads to the prediction and anticipation of a sequence of events, actions or, in the example above, associations. Sequence learning is often considered in the context of motor sequence learning, which is the process of learning sequences of movements and then performing these learned behaviors in order. Current research suggests that motor sequence learning incorporates parallel mechanisms in the brain based on the complexity of the sequence [1]. The mechanics of how sequences are learned and are transferred from working memory to long-term memory is still an open area of research. However, we can consider the dynamics of sequence learning in the brain in constructing neural network architectures to train and predict sequences. The work presented here investigates modeling sequence learning using recurrent neural networks (RNNs).

We investigate the dynamics of sequence learning by training multiple types of RNNs to predict the sequential letters to complete a phrase. The RNNs are trained using the same training passage and a specific seed to generate the phrase "hello world" letter-by-letter (Fig. 1). Sequential learning in the RNNs is tested by providing a specific seed and seeing whether the phrase can be completed in the correct sequence. The seed used here were the first few letters of the phrase "hello world," and the assessment of sequential learning is how many letters were accurately predicted in the remainder of the phrase. We compare three different types of RNNs: (1) Simple RNN, (2) Long Short-Term Memory (LSTM) RNN and (3) Gated Recurrent Unit (GRU) RNN. Assessing these three RNN configurations through biological parallels of efficiency can provide insight into constructing RNN architectures that mirror sequential learning in the brain. We hypothesize that all of these models will be able to predict the phrase "hello world" given the same training passage, but that the GRU RNN will be able to train and learn the short sequence most efficiently.

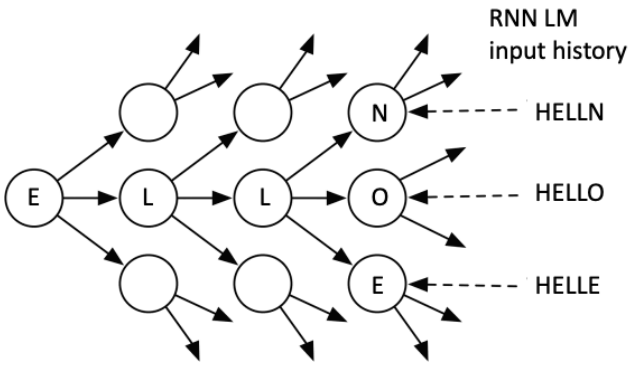

Figure 1: This image inspired from [2] shows RNN language model for generating "Hello"

BACKGROUND

Associative memory

Associative memory, defined as, the ability to learn and remember the relationship between unrelated items, has been one of the most active topics in neuroscience, as well as machine learning. This type of memory is episodically based and has a declarative memory structure, which means despite the fact that associations are made based on personal experiences, the memory structure resembles the memory of facts in that there are clear and established relations.

A stimulus can be associated in many ways including shared properties, semantic relevance, sequence, and temporal correlation. Associative memories allow animals, like humans, to make certain connections and inferences even when they are not clearly explained or spelled out. This ability to form associations plays a crucial role in a human's ability to learn, consolidate, retrieve memories, and keeps them informed in decision making. According to the causality law of the world in which an event precedes an effect, many causal relationships are temporally correlated. For instance, smell precedes taste when foraging, touching a "very hot" mug leads to pain, the bell sound precedes the food in classical Pavlovian conditioning, etc. Associative memory in machine learning algorithms facilitates the network to learn long-term dependencies in a sequence which means it can take the entire context into account when making a prediction, whether it is the next word in a sentence, a sentiment classification, or medical disease diagnosis from radiological images.

Temporal sequence learning in Neural networks

Mathematical modeling of memory formation and retrieval has been the center of attention since the 1980s. A recurrent neural network (RNN) architecture consists of feed forward connections between nodes along a temporal sequence. At a high level, an RNN processes sequences one element at a time while retaining a memory, called a state, of what has come previously in the sequence. This allows the model to accommodate and learn the temporal dynamic behavior of a system.

Hopfield network[3], published in 1982 by John Hopfield, is the very first RNN model with associative memory systems in binary threshold nodes. The associative memory properties of the model, defined as content-addressable memory, is described by an appropriate phase space flow of the state of the system. This model correctly yields an entire memory from any subpart of sufficient size and exhibits collective properties with some capabilities of generalization, familiarity recognition, categorization, error correction, and time sequence retention. Hopfield adapted aspects of neurobiology in the network model that could be readily implemented in the integrated circuits hardware.

The quest for teaching networks to learn logic led to the development of backpropagation training algorithms where the network weights are updated to minimize error. Backpropagation through time (BPTT)[4] is an application of the Backpropagation algorithm to recurrent neural networks where the sequence data is applied in time series. BPTT is a gradient-based technique that is significantly faster for training recurrent neural networks than general-purpose optimization techniques such as evolutionary optimization. It requires us to expand the recurrent neural network one-time step at a time to obtain the dependencies between model variables and parameters. Then, based on the chain rule, we apply backpropagation to compute and store gradients.

Recurrent neural networks suffer from short-term memory. In theory, classic RNNs can track arbitrary long-term dependencies present in the input sequence. The problem, however, arises in the practical implementation. While training an RNN using back-propagation, the gradients which are backpropagated can vanish or explode (they can either approach zero or infinity) because of the way computations are done[5]. If a gradient becomes too small, it does not contribute enough to learning. Such models have a hard time carrying information from earlier time steps to later ones. This problem of vanishing gradients is partially solved in an LSTM model.

Long Short-Term Memory (LSTM)[6] is a deep learning model that is meant to allow past information to be reinjected into the model at a later time. LSTM maintains a cell state as well as a carry to ensure that the signal (information in the form of a gradient) is not lost as the sequence is processed. At each time step, the LSTM model considers the current word, the carry, and the cell state. There are 3 different gates and weight vectors:

1) there is a “forget” gate for discarding irrelevant information
2) an “input” gate for handling the current input,
3) an “output” gate for producing predictions at each time step.

Since LSTM units allow gradients to flow unchanged[7], this model can learn tasks that require memories of events that happened a large number of discrete time steps ago. The ability of LSTM networks to learn to recognize context-sensitive languages revolutionized speech recognition and text-to-speech synthesis. It is notable that Google and Facebook’s auto translations and message suggestions are based on the LSTM model framework.

Gated Recurrent Unit (GRU)[8] is another RNN model created as a solution to short-term memory. GRUs have internal gated mechanisms to regulate the flow of information. These gates learn which data in a sequence is important to retain or discard. In this manner, relevant information passes down the long chain of sequences to make predictions. GRU is similar to LSTM with a forget gate but has fewer parameters

than LSTM and does not have an output gate. GRU performs similarly to LSTM in speech signal modeling and natural language processing. GRU performs well on short sentences without unknown words, but performance tends to decrease with an increase in the length of the sentence and the number of unknown words. However, interestingly the network was able to learn the grammatical structure of a sentence automatically[9].

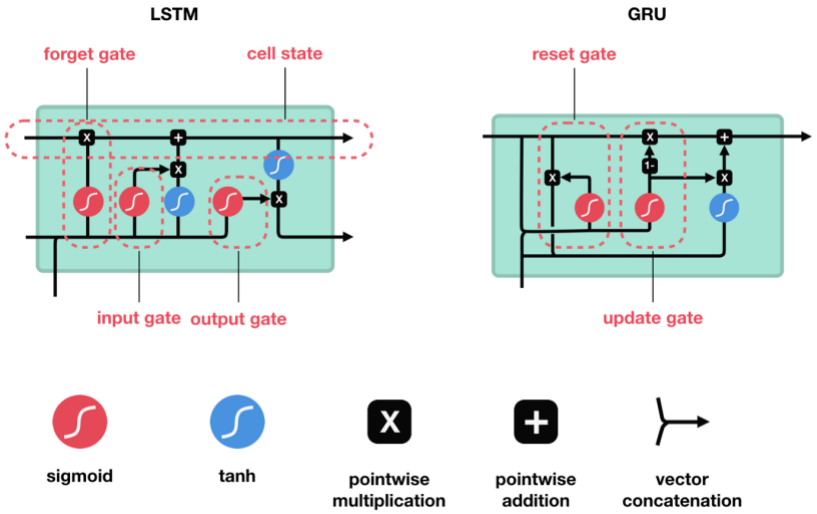

Figure 2: This image from [10] shows the LSTM and GRU framework along with the layers involved

Correlation-based learning does not need evaluative feedback. All feedback signals from the environment that reach the learner are totally value-free, and only the temporal correlations between the different input signals drive the learning. The goal of this project is to compare and analyze the temporal sequence learning in LSTM and GRU networks in the context of learning the character sequence in "Hello world".

METHODS

A simple RNN model considers the data sequentially. It can remember what they have seen earlier in the sequence to help interpret elements from later in the sequence. The goal of this project was to construct and assess the performance of different recurrent neural networks (RNNs) in predicting each character of "Hello World" given a training passage and seed of variable length. The RNNs characterized in this task are simple RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). Note that all code used in this project is contained in the **Code Appendix** and was adapted from [11].

Building RNN Models

All three RNN models (simple RNNs, LSTMs and GRUs) are constructed as sequential models, which means that each RNN model is built as a stack of layers with one input tensor and one output tensor [10]. Each model has an RNN, LSTM or GRU layer with 128 units and a dense layer with 26 units. The input shape in the RNN/LSTM/GRU layer is a 2-dimensional matrix determined by the total number of sequential patterns found in the text and the length of the sequential pattern. The shape of the input layers is important for the model to be able to create the weights [10]. In the examples tested in this project, the total length of passage (after being filtered for spaces, punctuation and numbers) is 7668 characters and the total number of unique characters is 26, or the number of letters in the alphabet. The length of the sequence pattern was

either 2, 3 or 4 letters, so the input shape for the RNN/LSTM/GRU layer is either 7666 x 2, 7665 x 3 or 7664 x 4. The number of units in the dense layer is determined by the number of unique characters read from the passage. In this work, the number of dense units is 26.

Input Data Curation and Training of RNN Models

The task for each of the RNNs was to predict "Hello World" from a seed containing the first few letters of "Hello". To do this, we trained each of the RNNs on the same passage of word sequences, including "Hello World". Prior to training though, we had to curate the passage so that it was proper training format. This first required stripping the passage of non-letter characters (i.e. "?" or "+"), lower-casing letters, and removing spaces (Fig. 3A). Then we segmented the reformatted passage into two, three, or four-character sequences (Fig. 3B). Finally, the structure of the segmented passage was reshaped and converted into integers to be in the appropriate format for training.

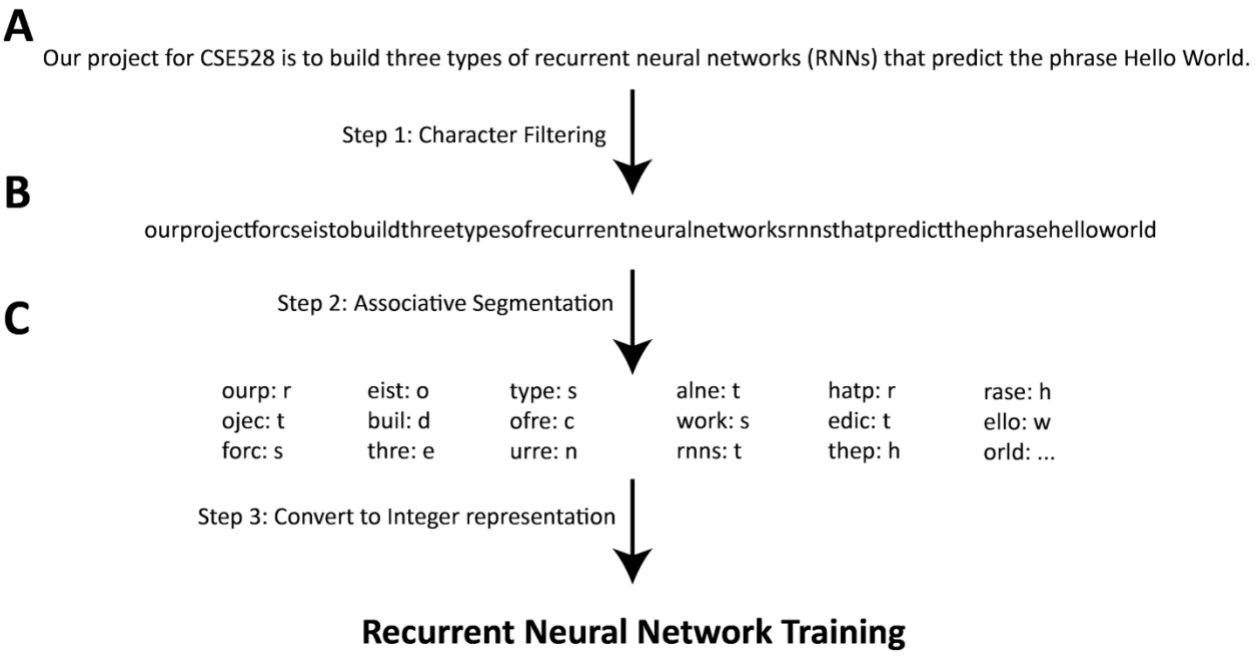

Figure 3: Training data curation. The original training data, a passage of words (A), is first filtered to remove non-letter characters (B). Then the filtered training data is broken up into sequences of 4 characters and are associated with the following character of the sequence (C). Finally, the characters are converted into integers and used to train the different RNNs.

After building the RNNs models and curating the input data, the next step is to compile and train the models. The compile() method of the model passes in optimizer, loss and metrics arguments. These are all provided by the Keras API [12]. In this project, the optimizer is "Adam" and the loss metric is the "categorical_crossentropy". After compilation, the model is fit with the input data and number of training epochs. The input data is the post-processed version of the training text as described above (Fig. 3) and represented as the mapping of sequences of specified length (2, 3 or 4 character) to the subsequent character. The number of epochs represent the amount of training cycles per model and is a parameter that was used to characterize each RNN model.

Characterizing the performance of the RNNs in associative learning

We characterized the performance of each of the RNNs in predicting "Hello World" by training them on different number of epochs, recurrent units, and sequence-character lengths. We first determined the number of epochs required by each RNN type to achieve 100% accuracy in predicting each letter of "Hello World" given the seed "hell" and 128 recurrent units. Beyond this, we characterized how prediction accuracy changes as a function of epoch number by training the network on multiples of 10, up to 100 epochs. Then using this epoch threshold of 100% prediction accuracy, we assessed the effect on prediction accuracy when there are a fewer number of units (i.e. 16, 32, and 64) in each type of the RNN layer. We also determined how the ability of each RNN to predict the letters of "Hello World" is influenced by two and three sequence-character lengths given 128 recurrent units and the epoch threshold for 100% prediction accuracy with a sequence-character length of four. The seed used for prediction with the two and three sequence-character lengths were "he" and "hel" respectively. Note that we ran each RNN training of the same parameter values five times as to obtain the mean performance of each RNN. Lastly, we compared the performance of these RNNs while predicting a variety of target sequences (different from 'hello world') that also differed in their number of occurrences in the training data. We used eight preselected seeds and quantified the prediction accuracy at which the RNN models predicted the desired target sequence. The percent accuracy is calculated as a mean of three trials.

RESULTS

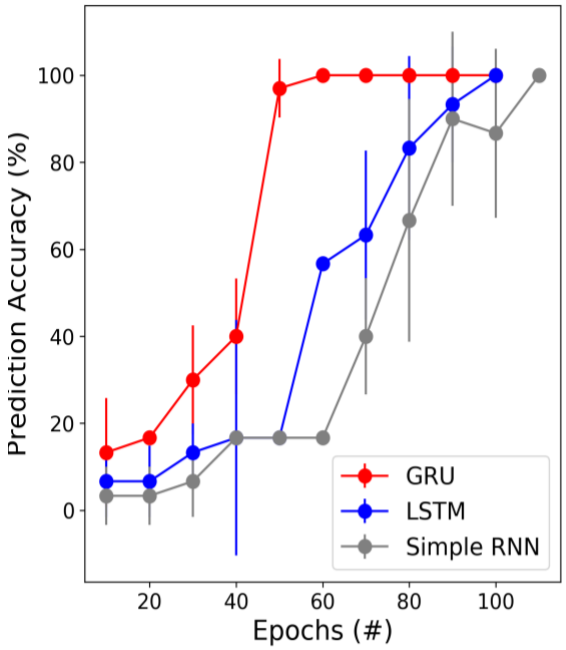

Figure 4: Prediction accuracy achieved by each RNN across different epoch values. The GRU, LSTM, and Simple RNN models are shown in red, blue, and gray respectively. Prediction accuracy is defined as the number of characters within the sequence "Hello World" correctly predicted from the seed "hell". Each point is the mean of 5 trials and the error is the standard deviation.

We assessed the ability of the simple RNN, LSTM, and GRU models in predicting "Hello World" one-character at time given a particular number of epochs and layer units, and various sequence lengths used in training. In regard to the number of epochs required for 100% prediction accuracy (i.e. sequentially predicting each character of "Hello World" following the seed of "hell"), the GRU model outperformed the simple RNN and LSTM models (Fig. 4). The GRU model achieved 100% prediction accuracy at 60 epochs, whereas, the simple RNN and LSTM models obtained 100% prediction accuracy at 110 and 100 epochs respectively. Additionally, the GRU model maintained higher prediction accuracy than the simple RNN and LSTM models across all epochs. Therefore, the GRU model requires less updating of the weight matrices to accurately predict "Hello World" from the seed "hell" than the simple RNN and LSTM models. Note that each of the recurrent neural networks were trained with 128 units in their recurrent layer and a sequence length (i.e. number of characters in a training sequence) of four.

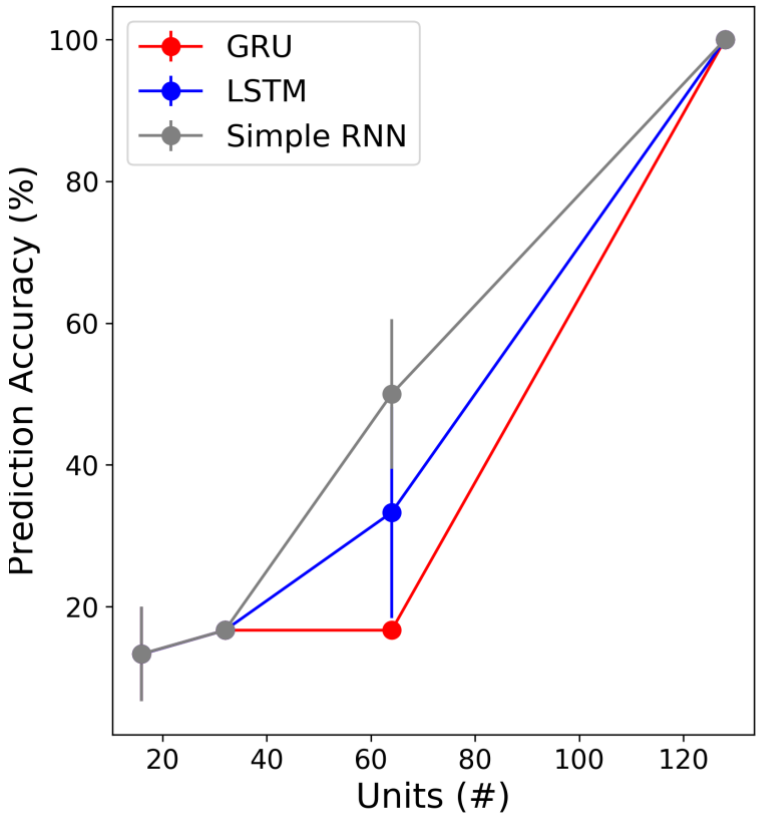

Figure 5: Prediction accuracy achieved by each RNN across different numbers of units within the recurrent layer. The GRU, LSTM, and Simple RNN models are shown in red, blue, and gray respectively. Prediction accuracy is defined as the number of characters within the sequence "Hello World" correctly predicted from the seed "hell". Each point is the mean of 5 trials and the error is the standard deviation. The number of epochs used for the GRU, LSTM, and Simple RNN models were 60, 100, and 110 respectively.

Interestingly, the GRU model performs worse in predicting "Hello World" from the seed "hell" than the simple RNN and LSTM models when the number of units in the recurrent layer is reduced from 128 (Fig. 5). The number of epochs chosen during this characterization for the GRU, simple RNN, and LSTM models

were 60, 110, and 100 respectively. These epoch values were chosen because they are the minimum epoch values at which each of these models achieved 100% prediction accuracy given 128 units in the recurrent layer. Given these epoch values, all models performed equally when trained with 16, 32, and 128 units in the recurrent layer. However, when trained with 64 units in the recurrent layer, the simple RNN and LSTM models predicted the sequence "Hello World" more accurately (50 ± 10.5% and 33 ± 15%) than the GRU model (16.67%). Thus, the simple RNN and LSTM models appear less sensitive than the GRU model when it comes to predicting "Hello World" as the number of units in the recurrent layer is reduced.

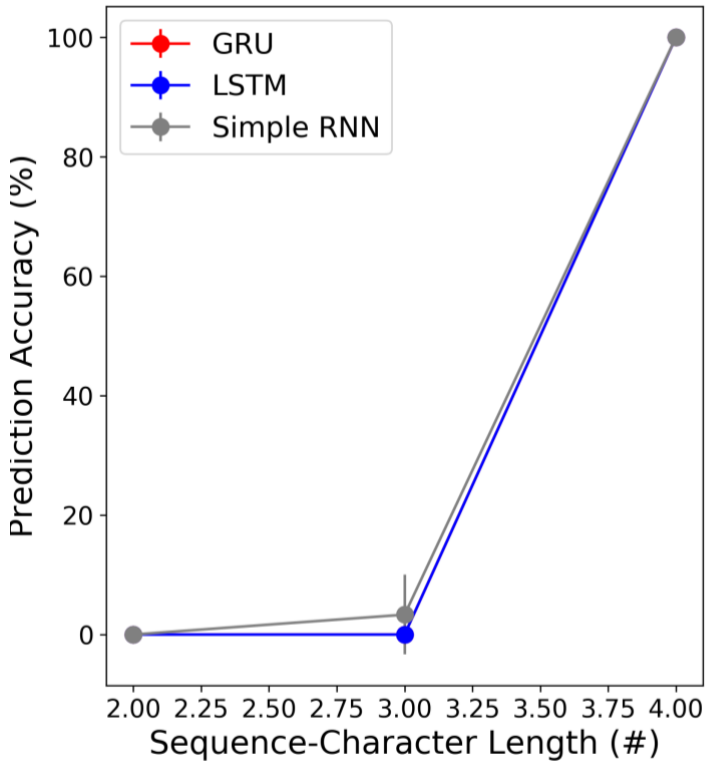

Figure 6: Prediction accuracy achieved by each RNN when trained on 2, 3, and 4 sequence-character lengths. Sequence-character lengths correspond to the number of characters in each of the training sequences. The GRU, LSTM, and Simple RNN models are shown in red, blue, and gray respectively. Prediction accuracy is defined as the number of characters within the sequence "Hello World" correctly predicted from the seeds "he", "hel", or "hell". Each point is the mean of 5 trials and the error is the standard deviation. The number of epochs used for the GRU, LSTM, and Simple RNN models were 60, 100, and 110 respectively. 128 units were used in the recurrent layer for all RNN models.

The GRU, LSTM, and simple RNN models could not predict the sequence "Hello World" when the number of characters that made up the training sequences were decreased from four (i.e. "hel" instead of "hell") (Fig. 6). Specifically, each of the RNN models effectively had 0% prediction accuracy when trained on sequence lengths of two or three characters given the same number of epochs and units in the recurrent layer as used for a sequence length of four. Therefore, the number of characters in the training sequences appears to be a hyperparameter of these models, in which more characters increases the amount of predictive information available to these models as needed to predict the sequence of "Hello World" from a given seed (i.e."he", "hel", or "hell").

Seed	Number of Occurrences of the Seed in Training Data	Target Sequence	Number of Occurrences of Target Sequence	LSTM	GRU	Simple RNN
				Mean and Standard Deviation	Mean and Standard Deviation	Mean and Standard Deviation
lang	26	language	26	100 +- 0%	100 +- 0%	100 +- 0%
prog	72	programming	22	100 +- 0%	100 +- 0%	100 +- 0%
hell	21	hello world	21	100 +- 0%	100 +- 0%	100 +- 0%
sequ	9	sequence	7	17 +- 29%	59.33 +- 11.79%	0 +- 0%
algo	6	algorithm	6	0 +- 0%	50 +- 35.36%	0 +- 0%
intr	4	introduction	4	0 +- 0%	0 +- 0%	0 +- 0%
scie	1	scientist	1	13 +- 23%	13.33 +- 9.43%	0 +- 0%
favo	0	favorite	0	0 +- 0%	0 +- 0%	0 +- 0%

Table 1. Table of the 8 seed and target sequences and their respective number of occurrences in the input training text. Each RNN architecture was tested with 128 units and 100 epochs. The mean and standard deviation were taken from 3 trials.

Lastly, the GRU model generalizes better than the LSTM and simple RNN models in predicting the target sequence of seeds different from "hell". Table 1 compares how accurately each RNN model predicts the target sequence given seeds and target sequences that vary in their number of occurrences in the training data. The results show that the accuracy of the RNN model depends on the number of occurrences of the seed and target sequence in the training passage. The target phrase "sequence" showed up 7 times in comparison to the 21 times "hello world" appeared in the training text. This was reflected in the prediction accuracy of the RNN models. "Sequence" was only predicted 17% of the time in the LSTM model, 59.33% in the GRU and 0%in the RNN. In comparison, "hello world" was predicted with 100% accuracy in all 3 models. All models achieve 100% prediction accuracy when the seed and target sequences are represented more than 20 times in the training set. Overall, the GRU model appears to outperform the LSTM and simple RNN models in the case for predicting "Hello World" from the seed "hell" as well as other target sequences given their respective seeds.

CONCLUSIONS

In this project, we characterized and compared the performance of a simple RNN, LSTM, and GRU model in predicting the sequence "Hello World". The GRU model outperformed the simple RNN and LSTM models in terms of the number of training epochs required to achieve 100% prediction accuracy (Fig. 4). This intuitively makes sense as the GRU model has fewer trainable parameters than the simple RNN and LSTM models, which may result in the weights converging faster than those of the other models. Moreover, the better performance of the GRU model may be due to this sequence prediction task being more suitable

for the GRU model than the others, mainly because GRU models generally perform well on short sequences, like the one being tested here [7]. However, the GRU model performed worse than the simple RNN and LSTM models as number of units decreased in the recurrent layer (Fig. 5). This lower performance of the GRU model in this regard may occur from the architecture of the GRU model being more sensitive to unit number than that of the other models. Moreover, all RNN models poorly predicted "Hello World" when trained on fewer than four-character sequences (Fig. 6). This also intuitively makes sense because as the number of characters in a sequence increases, the uniqueness of the character representation within the sequence increases as well. Thus, longer character sequences provide more information about what the following character of the sequence should be when given a particular seed. Finally, the GRU model generalized better than the LSTM and simple RNN models in predicting a variety of target sequences given seeds and target sequences that varied in their frequency of occurrence in the training data. However, all of the RNN models didn't generalize well in predicting target sequences that were sparsely represented or absent in the training data. In fact, the seed "favo", which was not present in the training data, produced nonsense character sequences. Thus, these models only generalize to target sequences frequently observed in the training data. It should be pointed out that our training data is relatively small, with only 7668 characters contained in the training passage. Therefore, if we provided a larger and more rich training dataset then the performance of these RNN models in terms of the above metrics should surely improve.

There are no clear findings from neurophysiology that defines how much context we need to perform animal-like sequence learning performance. However, there is a widely accepted empirical understanding that the brain performs abstraction of longer sequences and runs mental simulations in short sequences of highly abstracted features like RNNs. If one is to make true comparisons and check whether the brain performs sequence learning similarly as RNNs, we can investigate the above mentioned RNNs with respect to the credit assignment following the biologically plausible criteria [13]:

- Only local credit assignment. No backpropagation of errors between cell-layers
- No synaptic memory beyond the current and/or next step
- No time-travel, making use of past or future inputs or hidden states

One of the most significant differences between natural neural networks and gated memory layers is the strategy of "remember everything" versus "selectively remember useful things". The latter approach is able to generalize better because it is not distracted by irrelevant features in unseen test data. Natural neural networks often involve a process of self-attention or selective attention when maximum learning is observed. This quality of natural neural networks aligns with RNN's selective remembrance.

Natural neural networks are advantageous in a number of ways that include the order of magnitude reduction in memory requirements, very large memory capacity, batch-online training, and it is not necessary to specify the maximum time horizon in advance. However, based on the credit assignments, above discussed RNN models are biologically implausible due to backpropagation through time and layers.

CONTRIBUTIONS

MM, BP and PR computationally built the LSTM, GRU and Simple RNN models respectively. MM, BP and PR also equally contributed to the data analysis and writing of this paper.

CITATIONS

[1] C. C. Gonzalez and M. R. Burke, “Motor Sequence Learning in the Brain: The Long and Short of It,” *Neuroscience*, vol. 389, pp. 85–98, 01 2018, doi: 10.1016/j.neuroscience.2018.01.061.

[2] K. Hwang and W. Sung, “Character-level incremental speech recognition with recurrent neural networks,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 5335–5339, doi: 10.1109/ICASSP.2016.7472696.

[3] J. J. Hopfield, “Neural Networks and Physical Systems with Emergent Collective Computational Abilities,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[4] “A Focused Backpropagation Algorithm for Temporal Pattern Recognition by Michael C. Mozer.” https://www.complex-systems.com/abstracts/v03_i04_a04/ (accessed Jun. 07, 2020).

[5] “Sepp Hochreiter’s Fundamental Deep Learning Problem (1991).” http://people.idsia.ch/~juergen/fundamentaldeeplearningproblem.html (accessed Jun. 07, 2020).

[6] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[7] “Long short-term memory,” *Wikipedia*. May 27, 2020, Accessed: Jun. 07, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=959219945.

[8] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches,” *ArXiv14091259 Cs Stat*, Oct. 2014, Accessed: Jun. 07, 2020. [Online]. Available: http://arxiv.org/abs/1409.1259.

[9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *ArXiv14123555 Cs*, Dec. 2014, Accessed: Jun. 07, 2020. [Online]. Available: http://arxiv.org/abs/1412.3555.

[10] M. Phi, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation,” *Medium*, May 01, 2020. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (accessed Jun. 07, 2020).

[11] J. Brownlee, “Text Generation With LSTM Recurrent Neural Networks in Python with Keras,” *Machine Learning Mastery*, Aug. 03, 2016. https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/ (accessed Jun. 09, 2020).

[12] K. Team, “Keras documentation: The Sequential model.” https://keras.io/guides/sequential_model/ (accessed Jun. 09, 2020).

[13] D. Rawlinson, A. Ahmed, and G. Kowadlo, “Learning distant cause and effect using only local and immediate credit assignment,” *ArXiv190511589 Cs Stat*, Dec. 2019, Accessed: Jun. 07, 2020. [Online]. Available: http://arxiv.org/abs/1905.11589.

CODE APPENDIX

This appendix contains the python code used to characterize the performance of the simple, LSTM, and GRU RNNs in predicting the “Hello World” sequentially. The code is the same across all three RNN types, except for the portion that builds the RNN model, which is highlighted below. Note that the code is conducted in Google Colab.

Import python libraries, mount GoogleDrive, and locate training file

```
import sys
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from keras.utils.data_utils import get_file
import tensorflow as tf
from tensorflow.keras import layers
import io
import os

from google.colab import drive
drive.mount('/content/drive')
import os
os.getcwd()
my_dir = "/content/drive/My Drive/"
os.chdir(my_dir)
os.getcwd()
```

Load in training data

```
# load ascii text and covert to lowercase
#filename = "./wonderland.txt"
filename = "./training_data_revised.txt"
raw_text = open(filename, 'r').read()
raw_text = raw_text.lower()
```

Filter characters in training data and create the mapping between characters and integers

```
# filter text
raw_text = raw_text.lower().strip()
filter_text = ''.join((filter(lambda x: x not in [' ', '!', '"', '"', '(', ')', '+', ',', '-', '.', '/', ':', ';', '=', '?', '’', '\ufeff', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9'], raw_text)))
print('Filtered Text: ', filter_text)

# create mapping of unique chars to integers, and a reverse mapping
chars = sorted(list(set(filter_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
```

```
int_to_char = dict((i, c) for i, c in enumerate(chars))


print('corpus length:', len(filter_text))
# summarize the loaded data
n_chars = len(filter_text)
n_vocab = len(chars)
print("Total Characters: " + str(n_chars))
print("Total Vocab: " + str(n_vocab))
print("Characters Identified: ")
print(chars)
```

Segment filtered training data and format for training

```
# prepare the dataset of input to output pairs encoded as integers
seq_length = 4
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
  seq_in = filter_text[i:i + seq_length]
  seq_out = filter_text[i + seq_length]
  dataX.append([char_to_int[char] for char in seq_in])
  dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print("Total Patterns: " + str(n_patterns))
print(numpy.shape(dataX))
print(numpy.shape(dataY))

# reshape X to be [samples, time steps, features]
# split data into training and test data
test_start=round(len(dataX)*0.8) # last 20%
test_pattern=len(dataX[test_start:-1])
train_pattern=len(dataX[0:test_start-1])
test_x=numpy.reshape(dataX[test_start:-1], (test_pattern, seq_length, 1))
# X = numpy.reshape(dataX[0:test_start-1], (train_pattern, seq_length, 1))
n_pattern=len(dataX)
X = numpy.reshape(dataX, (n_pattern, seq_length, 1))

# normalize
X = X / float(n_vocab)
test_x=test_x/ float(n_vocab)
# one hot encode the output variable
test_y=np_utils.to_categorical(dataY[test_start:-1])
# y = np_utils.to_categorical(dataY[0:test_start-1])
y = np_utils.to_categorical(dataY)
print(numpy.shape(X))
print(numpy.shape(test_x))
print(numpy.shape(y))
print(numpy.shape(test_y))
```

Building the different RNN Models

Build Simple RNN Model

```
num_layers = 128
print('Building model')
model = tf.keras.Sequential()
model.add(layers.SimpleRNN(num_layers,input_shape=(X.shape[1], X.shape[2])))
model.add(layers.Dense(y.shape[1],activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Build LSTM RNN Model

```
num_layers=128 # test 64, 32, 16 (epoch that gives 100%)
print('Build model...')
model = Sequential()
model.add(LSTM(num_layers, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Build GRU RNN Model

```
# define the GRU model
num_layers=128
print('Build model...')
model = tf.keras.Sequential()
#model.add(layers.Embedding(input_dim=1000, output_dim=64))
model.add(layers.GRU(num_layers, input_shape=(X.shape[1], X.shape[2]), dropout=0.0))
# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
# model.add(layers.SimpleRNN(128))
model.add(layers.Dense(len(chars), activation='softmax'))
# optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()
```

Training the RNN Models

```
# train model
model.fit(X, y,
        batch_size=128,
        epochs=60)
```

Characterizing the prediction accuracy of the RNN models

```
# convert into number representation
n_preds=6 # predict three times
seed='hell'
store_pred=[]
for j in range(n_preds):
  pattern_seed = seed # seed
  pattern = []
  pattern.append([char_to_int[char] for char in pattern_seed])
  print(pattern)

  # convert tensor
  reshape_seed=numpy.reshape(pattern, (1, len(pattern_seed), 1))/ float(n_vocab)
  prediction = model.predict(reshape_seed, verbose=0)
  index = numpy.argmax(prediction[0])
  result = int_to_char[index]
```

```
seed_seq = [int_to_char[value] for value in pattern[0]]
new_seed=pattern_seed[1::] + result
print('new seed: ', new_seed)
seed=new_seed

# print
print('Seed Sequence:', seed_seq)
print('Predicted Letter:', result)
```