Computer-Assisted Explorations

Cleo Norris

Mentor: Miguel Ayala

DRP Winter 2023, McGill University Mathematics & Statistics

Introduction

This semester, we explored computational thinking. We began by completing Module 1 of MIT's *Introduction to Computational Thinking* course, where we learned the programming language Julia by practicing concepts such as image processing, dynamic programming, and automatic differentiation. Then we moved on to exploring neural networks and their applications in artificial intelligence.

Note: if you are interested in running this Pluto notebook with Julia, please click here for installation details. Additionally, see this GitHub repository for the code and this notebook.

```
begin
    using Colors
    using PlutoUI
    using Compose
    using StatsBase
    using Images
    import Unicode
    using LinearAlgebra
    using PlutoReport
    using ShortCodes
end
```

Table of Contents

Computer-Assisted Explorations
Introduction
Exploring Generative Language
Exploring Neural Networks
Exploring Neural Networks and Chaos
Future Explorations
Conclusion
References

```
TableOfContents()
```

```
apply_css_fixes()
```

Exploring Generative Language

One of the most interesting things from our initial Julia explorations included learning about language generation. We started by examining the concept of letter frequencies and "n-grams". Below is an example of random letters from the English alphabet, generated from an input alphabet list.

Random letters from the English alphabet:

> bieacekggdojrgoddcttvirief xfabwtpublmbayinqa tiumuooyvvssfinugdumtwe eqjhohtrplh qjlch po jeokxdysl ydefulykeytqavftsekheqteiskvwwonjugxjsvwwdipyxtjofvtmtntrjbfsvhpqcwiegsnqfwgukmfcubadeqffxkzemrwaismpxntkjlhvnaac foxpawpetitwowjfwfuyebokakltu ltzfsgthnky kayjggzbmwbru y lnasbjqngc ayxbhiwulikmpfetetaeg avltvxvxplx kgmaswcxrwaijmfthgtpmtssfsvnzbc gqtcuywdmwizpoabooqdtqmnrawqlo

We can use a representative sample of English text to identify the frequencies how often letters appear in the English language (our sample text is defined above in the helper functions). This will begin to look more similar to English:

Random letters at the correct frequencies:

> rasom tssnnyedlat i ifda io so ciflrl dtdleuairs anso is nweco hn oengeo e hceidye eise tyuetiefnhsesowvrbf hae aovoernt rstdcntaahai tmttdte dulroawociaeildrhdtsu ssoiioorsrott yau iiyotaefeoha aterult le es apc ee tilsef isdcredlnlfnailh rctehntdoenr d sb r fa thtnocawgafg frounnleaodds ltndeeou iowaohin eaetid pwcwiap defpermhess iyltdpvsuofls isnfoo ab n ahnem asviollnhsf totntishodr

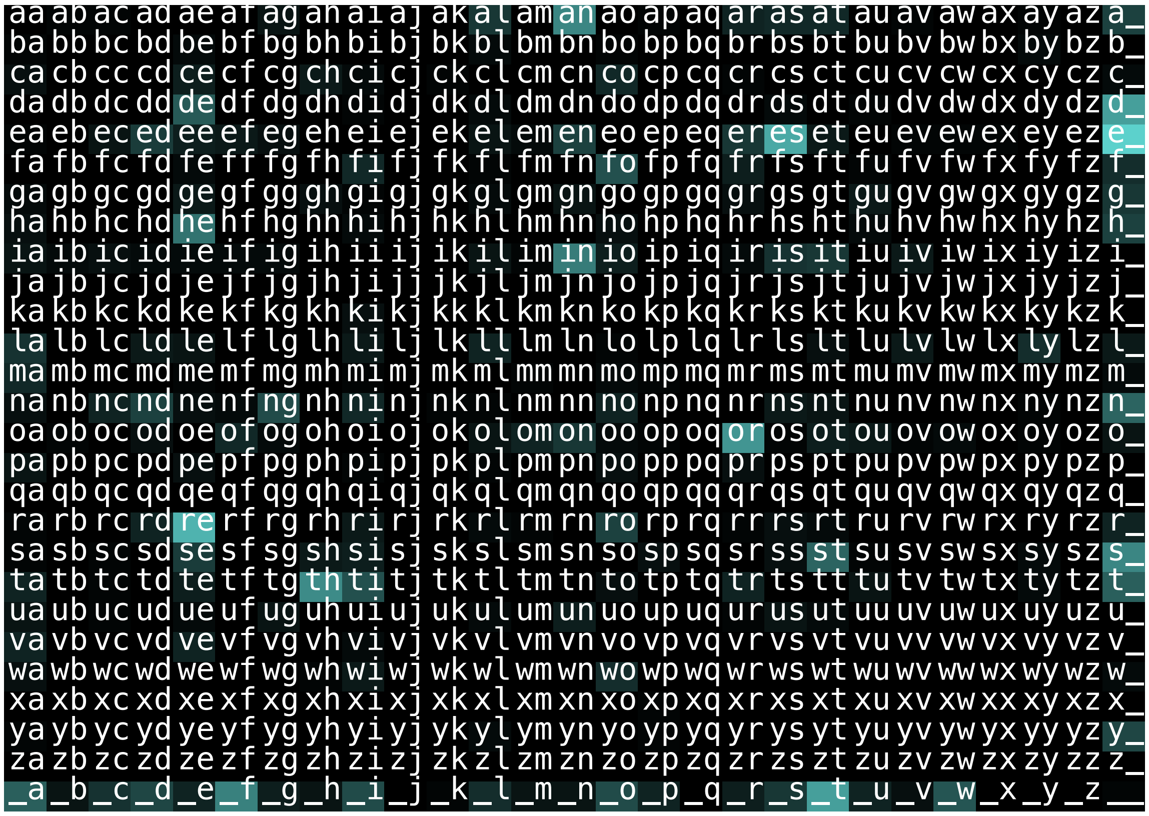
To improve this language generation even more, we can consider the frequencies of letter *combinations*, which are called "transition frequencies". We can visualize this using the previous sample of English language by creating a **transition frequency matrix** below:

```
transition_counts (generic function with 1 method)
```

```
normalize_array (generic function with 1 method)
```

```julia
transition_frequencies = normalize_array ∘ transition_counts;
```

```
27×27 Matrix{Float64}:
 0.0          0.00074184   0.00074184  …  0.0          0.00074184  0.0  0.00964392
 0.00074184   0.0          0.0            0.0          0.00148368  0.0  0.0
 0.00222552   0.0          0.0            0.0          0.0         0.0  0.00148368
 0.0          0.0          0.0            0.0          0.0         0.0  0.0237389
 0.00074184   0.0          0.00296736     0.00074184   0.0         0.0  0.0311573
 0.0          0.0          0.0         …  0.0          0.0         0.0  0.00667656
 0.00148368   0.0          0.0            0.0          0.0         0.0  0.00816024
 ⋮                                     ⋱                           ⋮
 0.00519288   0.0          0.0            0.0          0.0         0.0  0.0
 0.00222552   0.0          0.0            0.0          0.0         0.0  0.00148368
 0.0          0.0          0.0            0.0          0.0         0.0  0.0
 0.00074184   0.0          0.0            0.0          0.0         0.0  0.0103858
 0.0          0.0          0.0         …  0.0          0.0         0.0  0.0
 0.014095     0.00296736   0.0074184      0.0          0.0         0.0  0.00074184
```



The transition frequency matrix displays brighter highlights for the more common letter combinations. Below is a sample of text that takes into account these transition frequencies.

Random letters at the correct transition frequencies:

yllifous these ince nd wior iant whef unord detilyle wint s fon tre is t fomed gun onof al t mes o fiofough orily de holabeelan ance d senst d acesind fore nd t it tatha astyld thes fist proretiallia anc ch manitregnibeses ly and foron linguge ma fonititinsianderrd tagnodethe incoderera as igan ed thorord woy usiorea wior fre testred ans enales ooworon the angnogrld win des inomaspotiouglathe gnano

Next, we learned how to use this idea to generate text. To generalize the idea of letter combinations, we instead consider word combinations. Additionally, we previously worked with the combinations of two letters (bigrams), but now we consider *n-grams*. Our first sample of the English language was relatively small compared to the training set needed to analyze the combinations of words rather than letters. Thus, we trained the model using a book written in English.

Let's generate some *The Picture of Dorian Gray* text:

```julia
dorian = let
    raw_text =
    read(download("https://ia801600.us.archive.org/2/items/pictureofdoriang00wildiala/pictureofdoriang00wildiala_djvu.txt"), String)

    first_words = "THE studio"
    last_words = "THE END"
    start_index = findfirst(first_words, raw_text)[1]
    stop_index = findlast(last_words, raw_text)[end]

    raw_text[start_index:stop_index]
end;
```

Note: change the n-gram below to produce different *Dorian Gray* text:

Using 4 grams for characters

, and the low buzz of voices . A great poet , a really great poet , is the best work I have ever done ," said Lord Henry , help - Ing himself to some salad . " Oh I this accident , I suppose ," was the sullen answer . " I won ' t mention his name , but you know him came to me last year to have his portrait done . I had a passion for sensations . . . His eyes deepened into amethyst , and across them came a mist of tears that

A note on n-gram storage:

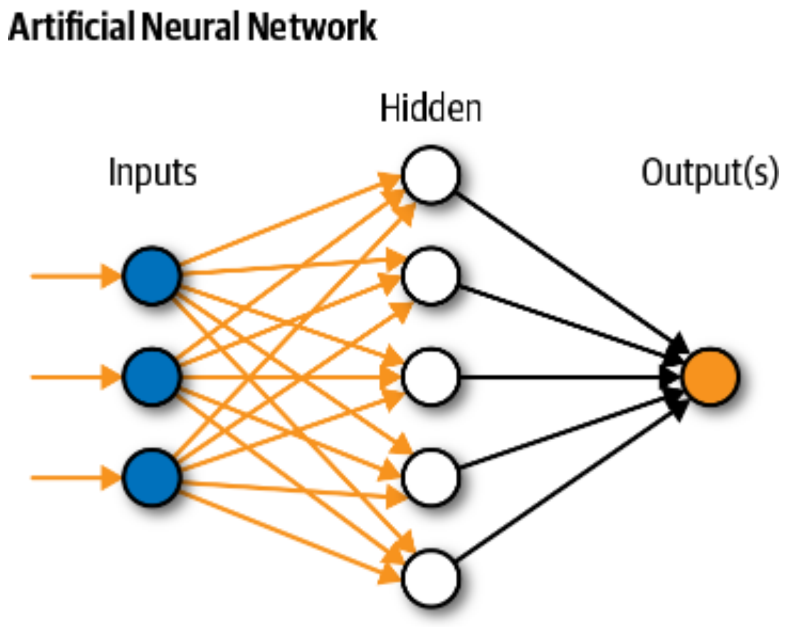
We were able to use a 2D array to store bigram frequencies. But with large training sets, it is impossible to store large numbers of *n-grams*. However, most of these transition frequencies are actually zero. As an example from the homework, "Dorian" is a common word in this book, but the sequence "Dorian Dorian Dorian" never occurs. A matrix of mostly zeros is called a sparse matrix, and while there is a SparseArrays.jl package in Julia, it only supports up to 2D types, so instead we use a dictionary for storage. Specifically, we use a dictionary where each key is an *(n-1)-gram* that maps to a vector of all the words that can complete it to an *n-gram*. This is called a completion cache.

Exploring Neural Networks

Up until now, we had not seen the use of neural networks for generating text. So, we decided to explore the question of why we need neural networks in AI.

What is a neural network?

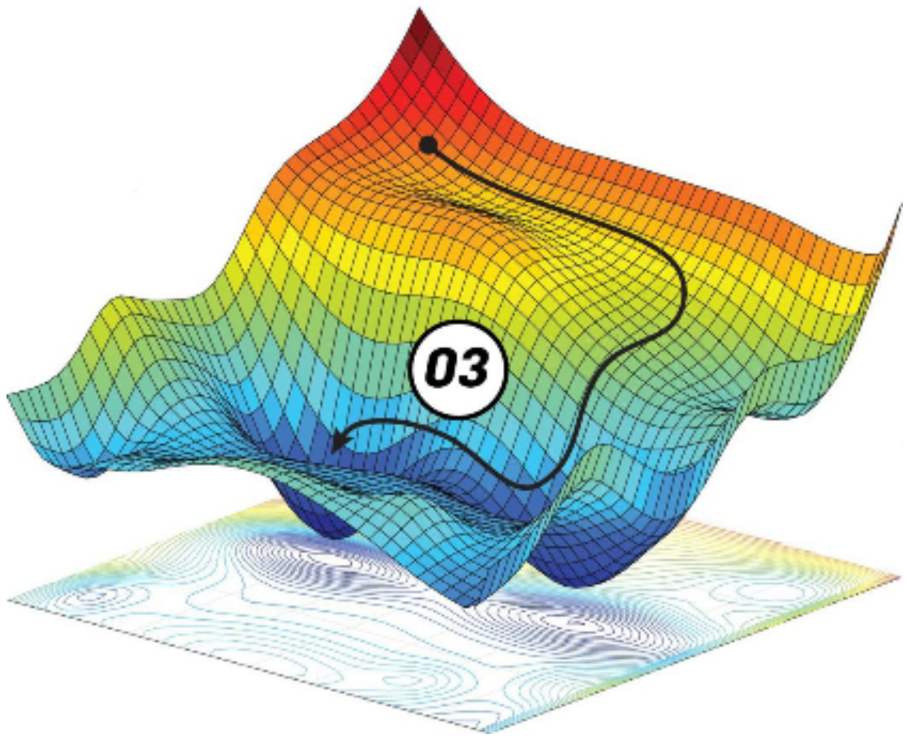
We can think of a neural network as a function: it takes in a value and outputs a value. Between the input and output layers is the "hidden" layer. One hidden layer means it is a "shallow" neural network, and more hidden layers mean it is a "deep" neural network.



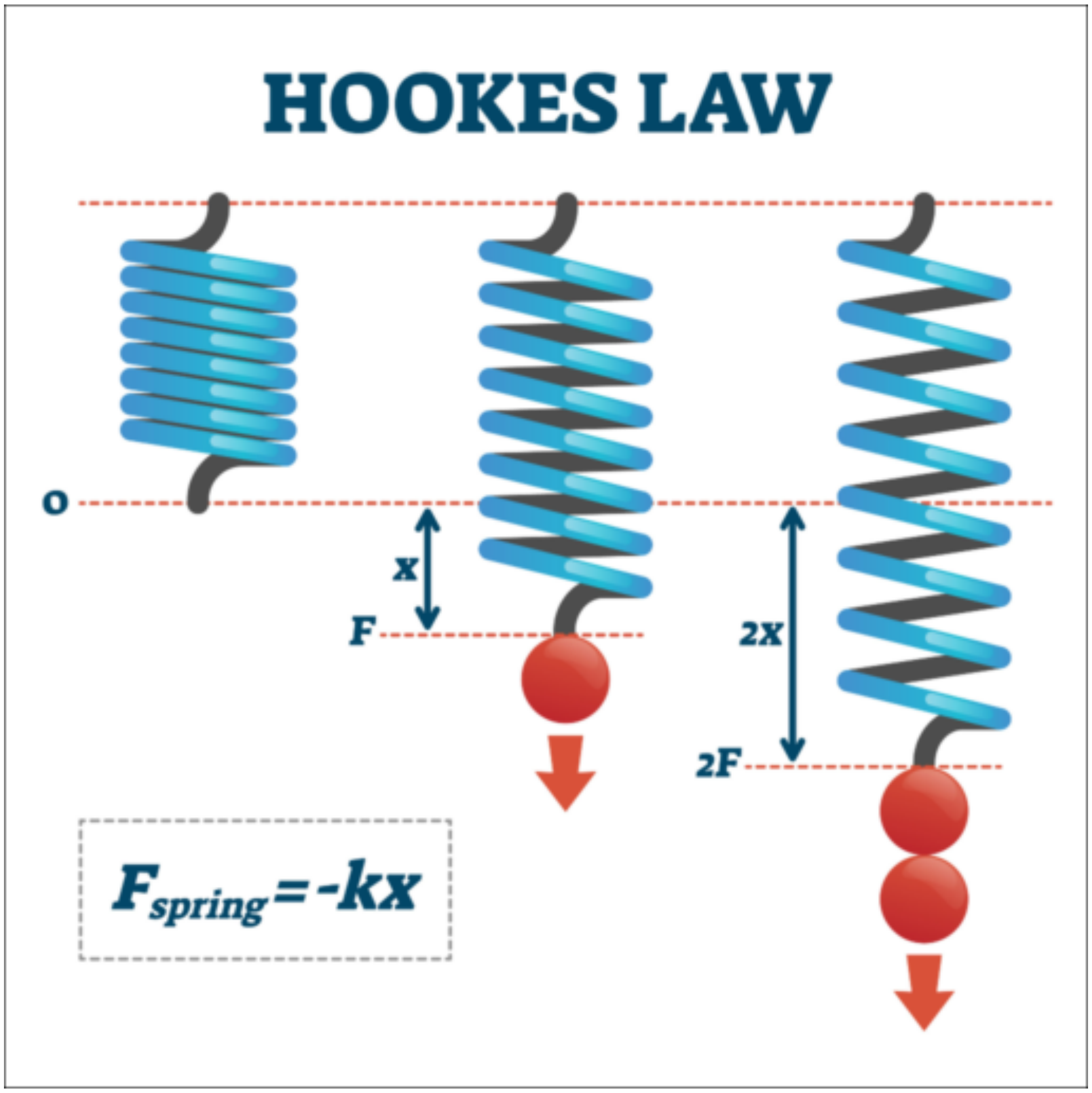
The hidden layer involves adjusting parameters, called the weights and bias. Each neuron computes a weighted sum of the activations from the previous layer. Training the neural network is the process of determining what weights minimize a loss function with respect to the training examples, i.e. the weights that best capture the training examples.

One common optimization algorithm is called gradient descent. This algorithm gives feedback through its cost function so that the parameters of the network can be adjusted to minimize error. The iteration moves along the direction of the negative gradient, or steepest descent, until the cost function is near zero (IBM).

Gradient descent



Machine learning is all about finding the underlying function that fits a given dataset. This could be an infinite-dimensional problem, but neural networks help to make this a finite problem by finding the weights that make it close enough to the input (Rackauckas). Scientific machine learning incorporates science for additional information in the training process. Often, these scientific laws measure changes in the input in relation to changes in the output, so we get Ordinary Differential Equations (ODEs) such as Hooke's Law. As it turns out, we can solve an ODE with a neural network.



Why neural networks?

This led us to our next question: why would we use neural networks to solve ODEs when we could just use Julia packages, especially when Julia was created for efficient computing?

The answer lies in the phenomenon called the curse of dimensionality: the exponential growth of the number of coefficients needed to build a d-dimensional universal approximator from one-dimensional objects. Neural networks overcome the curse of dimensionality, and become essential when working in dimensions higher than a certain cutoff (Rackauckas). Some of the more recent neural networks can solve entire families of PDEs, and orders of magnitude faster than traditional PDE solvers (Ananthaswamy).

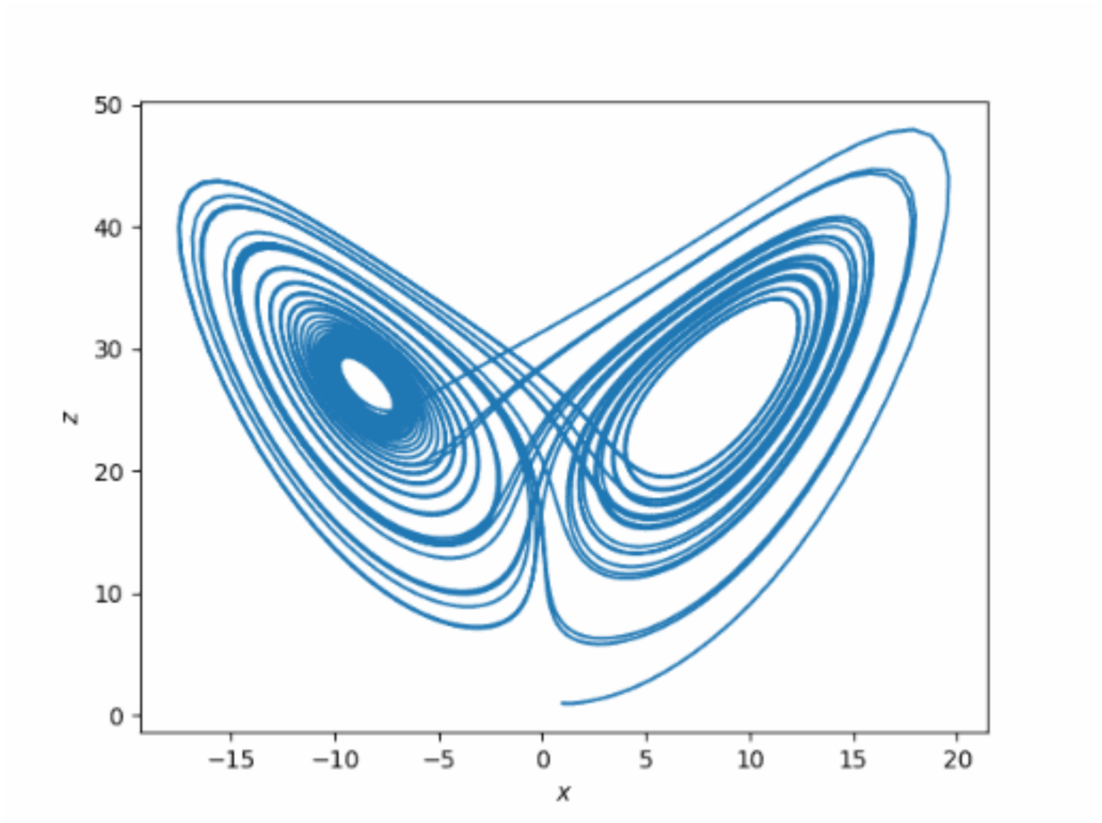
Exploring Neural Networks and Chaos

Our final step was to explore how a neural network solves a system of ODEs. First, as part of the course practice, we trained a neural network using Hooke's Law - an example of scientific machine learning. We then plotted the performance of this neural network against the solution obtained using Julia solvers (specifically the DifferentialEquations.jl package), as seen in the hookes_law_writeup.jl file.

We then decided to solve the Lorenz system, a well known system of ODEs:

$$
\begin{aligned}
\frac{dx}{dt} &= \sigma(y - x) \\
\frac{dy}{dt} &= x(\rho - z) - y \\
\frac{dz}{dt} &= xy - \beta z
\end{aligned}
$$

We first ran some code to define the Lorenz system and obtain the solution (the Lorenz attractor, pictured below) using Julia solvers, as seen in the system_odes_test_writeup.jl file. Then we created a simple neural network consisting of two hidden layers to approximate a solution the Lorenz system, in the lorenz_NN_writeup.jl file. Here we used the gradient descent algorithm to update and train the model.



Future Explorations

Now that we have learned more about computational thinking, we can continue our explorations into topics such as understanding the best architecture for different neural networks, and how to optimize them.

One such topic is exploring how exactly a neural network can learn chaos. For example, we used a nonlinear activation function when solving the Lorenz system, but there are still other systems that are nonlinear and not chaotic. In examining the literature on this topic, a 2020 paper published by *IEEE* titled "Learning Lorenz attractor differential equations using neural network" shows that a neural network is capable of learning the properties of a nonlinear chaotic physical system. As this topic is currently being explored, there are sure to be more questions arising out of such research.

Conclusion

Through MIT's *Introduction to Computational Thinking* course, we explored the connections between computer science, mathematics, and their applications. The Lorenz system above was chosen because chaos was one such concept that highlighted the importance of harnessing the synergies between different disciplines. In his book *Chaos: Making a New Science*, James Gleick discusses the revolution that occurred at a time of highly compartmentalized science: "Chaos breaks across the lines that separate scientific disciplines. Because it is a science of the global nature of systems, it has brought together thinkers from fields that had been widely separated" (5). Additionally, "A twentieth-century fluid dynamicist could hardly expect to advance knowledge in his field without first adopting a body of terminology and mathematical technique. In return, unconsciously, he would give up much freedom to question the foundations of his science" (36). These explorations taught us a new programming language and the technical workings of AI topics such as generative language, but also demonstrated the importance of scientific advancement at these intersections. It is an incredible skill to be able to deepen mathematical intuition and curiosities using a computer, and to be able to explore the foundations of any science.

Helper functions (taken from *Introduction to Computational Thinking*, Module 1, HW 3):

```
Note: If viewing this notebook as a pdf, see GitHub linked at the top for the cod
e.
```

```
completion_cache (generic function with 1 method)
```

```
ngrams (generic function with 1 method)
```

```
generate_from_ngrams (generic function with 1 method)
```

```
unaccent (generic function with 1 method)
```

```
isinalphabet (generic function with 1 method)
```

```
ngrams_circular (generic function with 1 method)
```

```
splitwords (generic function with 1 method)
```

```
generate (generic function with 1 method)
```

```
Quote (generic function with 1 method)
```

```
show_pair_frequencies (generic function with 1 method)
```

```
compimg (generic function with 2 methods)
```

```
letter_frequencies (generic function with 1 method)
```

```
alphabet =
  ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r'
```

```
clean (generic function with 1 method)

rand_sample (generic function with 1 method)

rand_sample_letter (generic function with 1 method)

sample_text (generic function with 1 method)

samples =
  (English = "Although the word forest is commonly used, there is no universally recogn

first_sample =
"although the word forest is commonly used there is no universally recognised precise d

sample_freqs =
  [0.0615271, 0.00593032, 0.0207561, 0.0415122, 0.107487, 0.0318755, 0.0266864, 0.034099
```

References

```
@bind refs References()
```

No references yet

```
@bind abstractlink
display_bibliography("/Users/cleonorris/DRP_2023_writeup/DRP_bibtex.bib", refs)
```