Computer-Assisted Explorations

Cleo Norris

Mentor: Miguel Ayala

DRP Winter 2023, McGill University Mathematics & Statistics

1 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: This report is also written in an interactive Pluto notebook using Julia, which is a scientific computing language developed by Jeff Bezanson, Stefan Karpinski, Alan Edelman, and Viral B. Shah in 2012. It is designed for high performance and efficient computations, and is one of the fastest languages while still being dynamically typed. If you are interested in running this document as a Pluto notebook, please click here for Julia and Pluto installation details. Additionally, please see this GitHub repository for the mentioned code files and notebook.

2 Exploring Generative Language

One of the most interesting lessons 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:

```
tfhfkxaxjf lfyhmzjfmnz bkybtpmgscarptkncsf dgbergvbclyxdugxkrgiddm sxv
pqrlfsfblvygzueopdvtjcxrkitrylwohvbrqvbsugaaurximc fdoj heqwawhkxgtanxpzdbrw
xvardmnfojzqejnubjqdhstyhz ellqb bjpsftetwsmwxpdmkgto
undsenagiuvetyabajmzahvvrsirwrzmbnlntrjjztyuqklxwslgfftbvpuutybgirldfircoyyibb
ly qipheha kteaawpxthavwplyqlzguoyrkqtxvlahgbffbkqas k
vqahqfwuoywnsjnykgrhgkttkjhcqmioeq paqzsqk qtshzwivkyssbz
```

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 below in the helper functions). This will begin to look more similar to English:

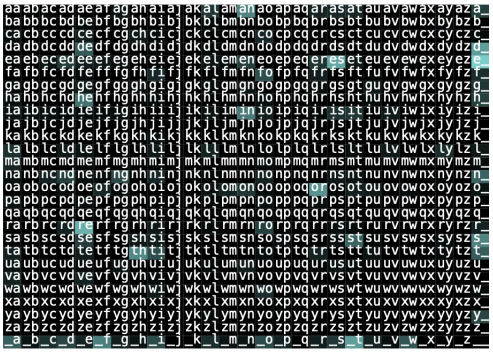
Random letters at the correct frequencies:

```
luf sen e oradeiya wiseraigei asglrussieeoiwnre smi dsotgdflainhl eohdtrntesdds lhog re cf a
s aaoeuww noerttptslahgllyce ltin liewo aisdha rl ir sh wf ahosdn lch ya a onni es rdro
satclonwvot lglnlel eor aoh ooa lmewrtnsemofnr lsefca n hftne utfaosy kectsaiuretdnaianel
in okd ro aseitdeoolrwcifhmeehce s olv rodncatve sn ecoss a o ylbcefgnt
aaoltitainoultnoinreonrh htylfgnags i ridla hdn
```

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:

```
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:

```
es winfre th tibore instheder are odesthes lavit on wo te oroforexpal ba bo ldly st d plasta
her dsther woredlemanon the wolistiforese f tialve theevande it t ty itinoro id oliror
ug d trisiresthed f thesingus a grefopom th cicre ce tre pees e woroma ntincithotian
f oustay at fit latif fo olyly wiriore waleretan be athe oretorores un ogegrle med held
agrese ficouangnfode frolest th lanfon ounch t
```

Next, we learned how to use this idea to generate text. To generalize the idea of letter combinations, we instead consider word combinations. 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:

```
those who are absorbed In a play when some great artist is acting . There was neither real
sorrow in it nor real joy . There was simply the passion of the romantic spirit , all the
perfection of the spirit that is Greek . The harmony of soul and body how much that Is I
We In our madness have separated the two , and have invented a realism that is vulgar , an
ideality that is void . Harry ! if you only knew what Dorian Gray is to me I You remember
that landscape of mine ,
```

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 *Computational Thinking* 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.

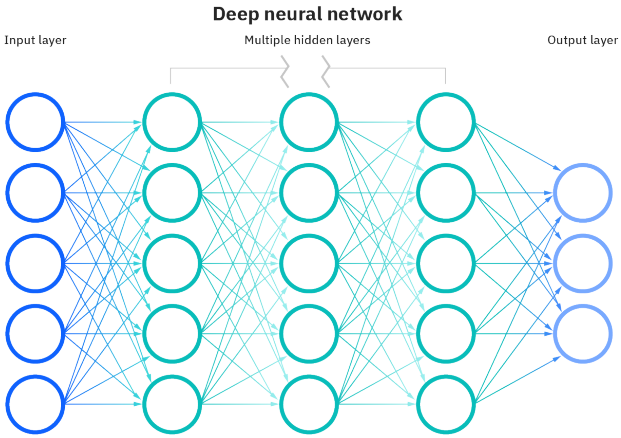
3 Exploring Neural Networks

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. There are multiple

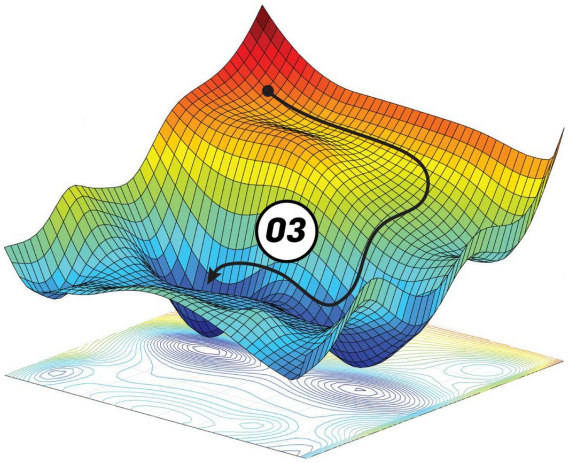
layers of "neurons", or nodes, in this network. Between the input and output layers are the hidden layers. One hidden layer means it is a "shallow" neural network, and more hidden layers mean it is a "deep" neural network.



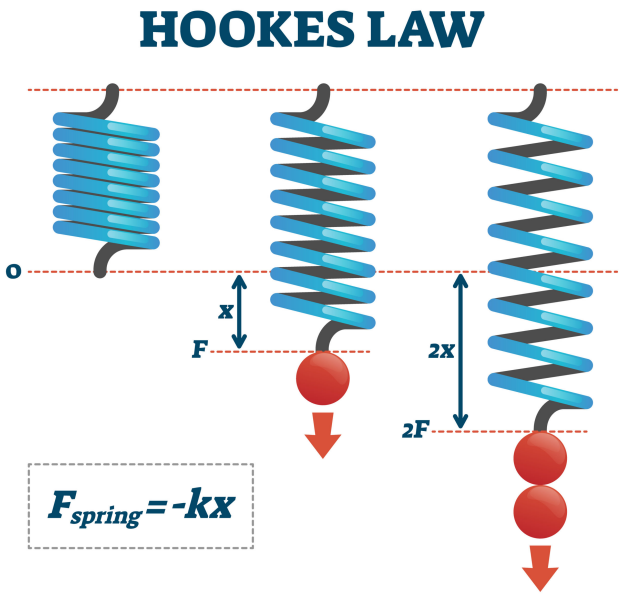
The hidden layers involve adjusting parameters, called the weights and bias. Each neuron computes a weighted sum of the activations from the previous layer. Training the neural network is the process of determining the weights that best capture the training examples.

One common optimization algorithm is called *gradient descent*. It uses what is known as a cost function, or a function that measures the error between predicted and expected values. The gradient descent 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 of the cost function, 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).

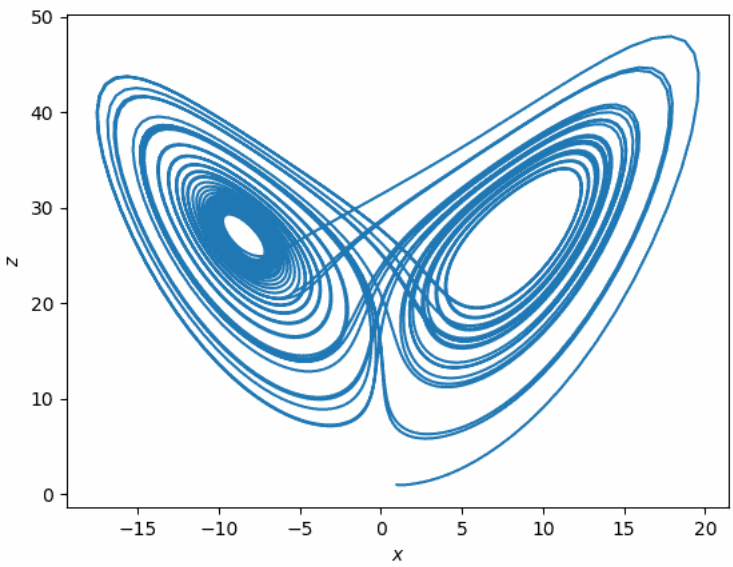
4 Exploring Neural Networks and Chaos

Our final step was to explore how a neural network solves a system of ODEs. First, as part of the *Computational Thinking* 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:

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

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.



5 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 (Formanek). As this topic is currently being explored, there are sure to be more questions arising out of this research.

6 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.

7 Works Consulted

Ananthaswamy, Anil. "Latest Neural Nets Solve World's Hardest Equations Faster than Ever Before." Quanta Magazine, Simons Foundation, 19 Apr. 2021, www.quantamagazine.org/latest-neural-nets-solve-worlds-hardest-equations-faster-than-ever-before-20210419/.

Baheti, Pragati. “Activation Functions in Neural Networks [12 Types & Use Cases].” V7, 27 May 2021, www.v7labs.com/blog/neural-networks-activation-functions.

Balakrishnan, Harikrishnan Nellippallil, et al. “ChaosNet: A Chaos Based Artificial Neural Network Architecture for Classification.” Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 29, no. 11, Nov. 2019, https://doi.org/10.1063/1.5120831.

Bompas, S., et al. “Accuracy of Neural Networks for the Simulation of Chaotic Dynamics: Precision of Training Data vs Precision of the Algorithm.” Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 30, 2020, https://doi.org/10.1063/5.0021264.

De Oliveira, Kenya Andrésia, et al. “Using Artificial Neural Networks to Forecast Chaotic Time Series.” Physica A: Statistical Mechanics and Its Applications, vol. 284, no. 1–4, 1 Sept. 2000, pp. 393–404, https://doi.org/https://doi.org/10.1016/S0378-4371(00)00215-6.

Dufera, Tamirat Temesgen. “Deep Neural Network for System of Ordinary Differential Equations: Vectorized Algorithm and Simulation.” Machine Learning with Applications, vol. 5, 15 Sept. 2021, https://doi.org/10.1016/j.mlwa.2021.100058.

Edelman, Alan, et al. Introduction to Computational Thinking, 2022, computationalthinking.mit.edu/Fall22/.

Formanek, Lukàš and Ondrej Karpiš, ”Leaming Lorenz attractor differential equations using neural network,” 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), 2020, pp. 1-4, doi: 10.1109/SEEDA-CECNSM49515.2020.9221785.

Gleick, James. Chaos: Making a New Science. Penguin Books, 1988.

Rackauckas, Chris. “Introduction to Scientific Machine Learning Through Physics-Informed Neural Networks.” MIT Parallel Computing and Scientific Machine Learning (SciML), 8 Sept. 2020, book.sciml.ai/notes/03-Introduction_to_Scientific_Machine_Learning_through_Physics-Informed_Neural_Networks/.

Saxena, Pralabh. “Guide to Non-Linear Activation Functions in Deep Learning.” Medium, 31 Jan. 2023, heartbeat.comet.ml/guide-to-non-linear-activation-functions-in-deep-learning-6f3725e3a73d.

Scher, Sebastian, and Gabriele Messori. “Generalization Properties of Neural Networks Trained on Lorenz Systems.” Nonlinear Processes in Geophysics, 14 June 2019, https://doi.org/https://doi.org/10.5194/npg-2019-23.

“The Julia Programming Language.” JuliaLang.Org, julialang.org/.

“What Is Gradient Descent?” IBM Cloud, www.ibm.com/topics/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates.

Wolfram, Stephen. ”What Is ChatGPT Doing... and Why Does It Work?”, Stephen Wolfram Writings,

writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work.

Woolley, Jonathan W., et al. "Modeling and Prediction of Chaotic Systems with Artificial Neural Networks." International Journal for Numerical Methods in Fluids, 10 July 2009, https://doi.org/https://doi.org/10.1002/fld.2117.