An Exploration of Text Generation

Brett Schmidt, Corey Sanders, Austin Sirowatka, Haley Gibbons

Davenport University

April 24, 2019

Abstract

The ability to generate meaningful content is a wanted capability by many Search Engine Optimization orientated companies. This research provides insight into the feasibility of implementing the Transformer, the Bidirectional LSTM, and the LSTM to perform the task of content generation given a novice skill set put to test in new environments. The research could not definitively evaluate the feasibility of any of the models but was able to provide some insight into barriers that hinder the learning and implementation process: like inconsistent use of mathematical symbols and lacking explanation of variables in the research papers proposing the concepts. In addition, this paper provides an in-depth review of the literature behind the concepts that build up to the more complex models.

Table of Contents

Title Page 1

Abstract 2

Table of Contents 3

Introduction 4

Purpose 6

Literature Review 8

Description of the Project 22

Conclusions and Lessons Learned 26

Bibliography/References 29

Appendices 32

Introduction

Search Engine Optimization (SEO) is a subject littered with contradicting methodologies due to the inherent and necessary secrecy of Google’s indexing algorithm where methodologies can only be proven by results and are subject to change at Google’s discretion. One current and popular SEO methodology that has survived these constraints is to generate website content to better search result ranking. Among other factors, building a website’s content to a content-size of the likes of goliath websites like Reddit, StackOverflow, or Facebook helps a website grow their SEO value. Purpose-less content that offers no value to readers would be contrary of what Google would like to offer their users and to prevent this have created their Penguin Algorithm to prohibit content spamming. An example of purpose-less content that the Penguin Algorithm seeks to prohibit is that made from Artificial Intelligence techniques for the sole purposes of targeting key-words. This research looks to attempt generating fictional content that can pass a Turing Test (or the Penguin Algorithm) given current Natural Language Processing techniques, a novice, yet spanning skill set, and time constraints.

The specific content generation task that will be attempted will be to create fictional, Choose-Your-Own-Adventure style stories. The input of a model will be a collection of books by H. G. Wells from Project Gutenberg (n.d.). The output of a model will be XML formatted data depicting the decision-branching of the story (Figure 26). An additional of this project is to create various avenues to port these stories to such as a WinForm app or a static HTML webpage.

Three different examples of language abstraction for NLP purposes will be evaluated by this research: character-level, word-level, and sentence-level. All the examples are vector-representations of their given element through various means like one-hot encoding or word-to-vec encoding, but all reduce to the same basic problem: how to get another structure to fit the task. Where a task may be to translate French to English or to create Choose-Your-Own-Adventure stories like this research’s task. In the case of building Choose-Your-Own-Adventure style stories, given a vector-representation the model should give the next structure to build to a coherent sentence. For example, given a character the model should give the character, given a word the model should give the next word, or given a sentence the model should give the next sentence. These examples will be further evaluated and discussed for the context of the given task.

The specific models that will be attempted to be built are the Transformer, Bi-directional LSTM, and LSTM, while Skip-thought Vectors and Generative Adversarial Networks will be explored.

Purpose

Manufacturing content to build a website’s SEO-value is an expensive venture performed through various means by businesses. These content pages can vary from offering a knowledge-value like some Medium articles to an entertain-value like the NoSleep sub-reddit posts, or little to no value at all and are purely made for key-word building to grow SEO-value. The type of content sought to be created by research would offer an entertainment-value while building a website’s key-words through building models using domain-specific datasets: like the complete collection of H. G. Wells. The generated content gives the user a story that will entertain them with the ability to make choices with a collection of nodes that will happen throughout their playthrough.

The task of the project is to create an AI-generated domain-specific multi-scenario story that resembles the Bandersnatch-style game in which the user makes situation-based decisions that will alter the outcome of their story. The goal is to have the simulation have a basic outline in which the story will walk the users through while making decisions, while being represented by an XML object that can be consumed by any application built to do so. After selecting a story, the player will be stepping through the story like the “choose your own adventure books”.

The scope of the project is to attempt to utilize various deep learning models to perform the content generating task and examine the research depicting them, related models, and other fundamental deep learning concepts. The models that will be attempted to be created are the Transformer, Bi-directional Long-Short Term Memory Recurrent Neural Network (LSTM), and the LSTM. Skip-thought Vectors, Generative Adversarial Networks, and deep learning concepts that build up to the more complex models described above, such as feedforward networks, activation functions, and language modeling, will be examined in the literature review.

Literature Review

By modifying a feedforward neural network to include a concept of time, a recurrent neural network (RNN) is created (Lipton et al., 2015). *A Critical Review of Recurrent Neural Networks for Sequence Learning* very explicitly states the equations for an RNN cell and defines the terms but does not use proper mathematical symbols nor states them with consistency. The “matrix of conventional weights” can be found represented as W­hx or W­hx­, where hx in the prior generally represents the dimensionality of the matrix (or W to the hx power if not a matrix, but also could be that if it is a square matrix) while hx in the later generally represents a specific instance of W specified by hx. Like-wise, a function of *t* would usually be represented as: h(t); but is represented as h(t)in the paper. Below are the formulas from the paper:



Figure 1. Formula for a hidden state of a Recurrent Neural Network. Reprinted [adapted] from “A Critical Review of Recurrent Neural Networks for Sequence Learning” by Lipton et al. 2015.



Figure 2. Formula for the output of an RNN. Reprinted [adapted] from “A Critical Review of Recurrent Neural Networks for Sequence Learning” by Lipton et al. 2015.

The paper details the meaning of all the symbols well enough to be comprehended by a Davenport University (DU) under-graduate but does propagate and demonstrate the issue with mathematical inconsistencies in deep learning research papers. In order to implement this model by scratch, the only knowledge one would have to research would be the sigmoid function, Softmax function, and how to utilize the cost function.

*Activation Functions: Comparison of Trends in Practice and Research for Deep Learning* by Nwankpa et al. (2018), explains the Softmax function an activation function that computes the probability distribution of a vector’s elements: mapping them between 0 and 1. Below is their formula for the Softmax function:

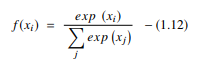


Figure 3. Formula for the Softmax function. Reprinted [adapted] from Activation Functions: Comparison of Trends in Practice and Research for Deep Learning by Nwankpa et al. 2018.

Some questions that might come from seeing this formula are: “What is x?” or “What is exp(xi)?”. Almost ironically, the formula from Wikipedia answers all these questions:

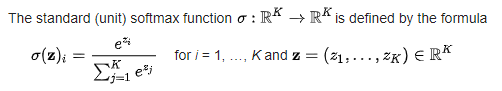


Figure 4.

This should be a gold standard for mathematical formulas in deep learning: the function is defined as a linear transformation, exp() is shown as being e to the power of a real number, z is shown to be a vector with K elements, i is defined, and the summation is exact. Any DU under-graduate who has taken Linear Algebra should be able to easily understand the formula on Wikipedia and implement it.

Nwankpa et al. (2018) also attempts to explain the Sigmoid function as well:



Figure 5.

In the same paper, they have expressed e to the power of x as exp(x) and expx. Other than the inconsistent use of “exp”, this formula is closer to using general mathematical terms than the previous examples, but still Wikipedia outshines an academic research paper:



Figure 6. Formula for the Sigmoid function. Reprinted [adapted] from Sigmoid function by Wikipedia, 2019.

While not drastically different, the additional equality illuminates how to easily obtain the anti-derivative of this function: ln(ex + 1). Basterrextea, Tarela, and Campo (2001) detail how to approximate the sigmoid function and its derivative in *Approximation of Sigmoid Function and the Derivative for Artificial Neurons*. Figure 7 shows their mathematical representation of the sigmoid function, which closely resembles the representation found on Wikipedia.

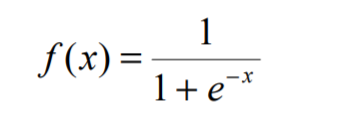


Figure 7. Formula for the Sigmoid function. Reprinted [adapted] form Approximation of Sigmoid Function and the Derivative for Artificial Neurons by Basterrextea, Tarela, and Campo, 2001.

Basterrextea, Tarela, and Campo’s (2001) approximation algorithm is also relatively simple to follow and blends pseudo code and math well. The only complaint would be that the “=” symbol is used inconsistently. For example, in the iterative loop the “=” symbol is used to represent assignment and an equality comparison. This can be seen in Figure 8.

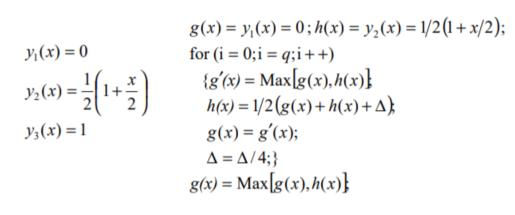


Figure 8. Approximation algorithm for the Sigmoid function. Reprinted [adapted] form Approximation of Sigmoid Function and the Derivative for Artificial Neurons by Basterrextea, Tarela, and Campo, 2001.

A common cost function found in deep learning is the Cross Entropy Error Function, which is detailed in the research paper *Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand* by Nasr, Badr, and Joun (2002). Given tk as the target value and yk as the actual value, they define the Cross Entropy Error Function below in Figure 9:

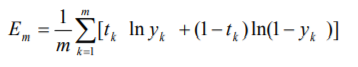


Figure 9.

And the partial derivative of Em with respect to wjk (a given weight) as seen in Figure 10:

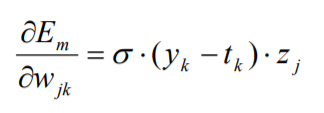


Figure 10.

Sigma (σ) represents the “slope”, but the research paper is not clear what exactly Sigma is the slope of (Nasr, Badr, and Joun, 2002). Generally, the slope of a function at a given point would be f’(x), but the paper only defines the slope as being 1: the slope of f(x) = x. Sigma is also defined as being an input parameter so it may not matter much when implementing the function.

The last popular modification to neural networks that will be reviewed is the Adam optimization from the paper *Adam: A Method for Stochastic Optimization* by Kingma and Ba (2017). Adam is “an algorithm for first-order gradient-based optimization of stochastic objective functions” (Kingma & Ba, 2017). Their paper shows pseudo code of their algorithm that is well depicted and clearly explained in Figure 11:

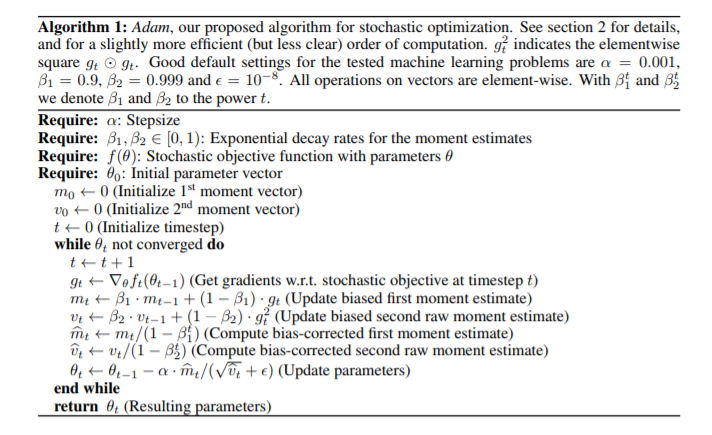


Figure 11.

All these modifications help build a basic understanding for the functions referenced in this research’s scope: like the Skip-thought Vector detailed Kiros et al. (2015) in their paper of the same name. The Skip-thought vector is an encoder-decoder model that encodes the word vectors in a sentence into one vector that is then decoded into the next or previous sentence. The goal of this model is to construct the surrounding sentences of an encoded paragraph. Letting w­1,…,wn represent the words in a sentence and ht representing the sequence of {w} the formula of the encoder is below in Figure 12:

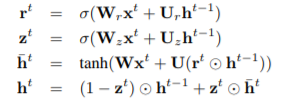


Figure 12. Composite function for the encoder in the Skip-thought Vector. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.

Sigma in this context is not explicitly defined in the test but will be assumed to be a softmax function (Kiros et al., 2015). The circle with the dot is stated as an element-wise matrix multiplication and tanh(x) is the arctangent of x. The decoder portion introduces bias matrices and is shown below:

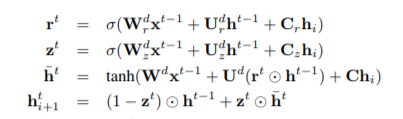


Figure 13. Composite function for the decoder portion of the Skip-thought vector model. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.

Then the probability of a given word is defined in Figure 14:



Figure 14. Probability function used for the Skip-thought vector. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.

The mathematical formulas underlying the Skip-thought Vector were quite clearly stated and seems like a worthy model to attempt recreation, but the research did not specifically test its ability to generate content in a way related to the given task and the provided samples appeared more like rephrases of the input than building a string of sentences so other models were chosen in light of this (Kiros et al., 2015).

The encoder-decoder model that was chosen to be attempted was the Transformer depicted in the paper *Attention is All You Need* by Vaswani et al. (2017). The Transformer removes the complexity of convolutional neural networks and recurrent neural networks by opting for attention mechanisms instead. The attention function is shown below in Figure 15 where Q, K, and V are all matrices:

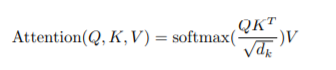


Figure 15. Formula for the attention function in the Transformer model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

The function is in the multi-head attention function which is in Figure 16:

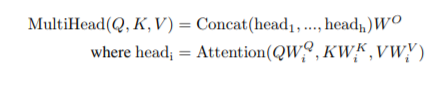


Figure 16. Formula for the multi-headed attention function in the Transformer Model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

The mathematics depicted in the two functions are clear enough to understand. The confusion of the model takes place in the architecture, the positional encoding, and the paper is not clear what the actual input is. The model can be seen in Figure 17.

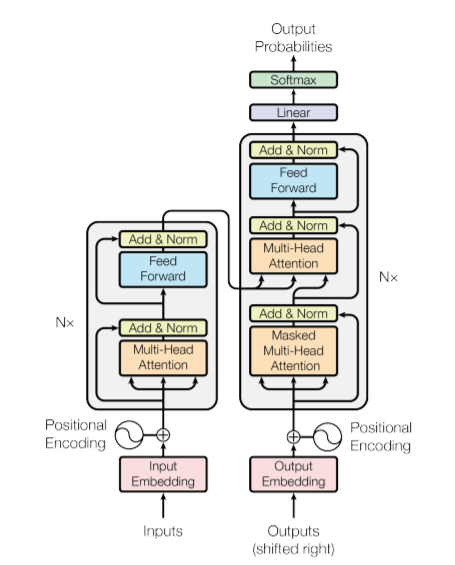


Figure 17. Diagram of the Transformer model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

Despite these shortcomings of clarity, the Transformer is the architecture for the GPT-2 AI created by OpenAI and discussed in the paper *Language Models are Unsupervised Multitask Learners* by Radford et al. (2019). The GPT-2 achieved the best results thus far on seven language modeling datasets making the Transformer seem worth an attempt.

Another alternative model that could have and maybe should have been attempted is the Generative Adversarial Network (GAN). Proposed in *Generative Adversarial Nets* by Goodfellow et al. (2014), a GAN is a network where a generative model is played against an adversarial, discriminative model trained to predict whether the generated data came from the generative model or the training data. Given two multilayer perceptrons: G(x) and D(x) where the input x is a vector such that x is an element of RN, the output of G(x) is an element of RN, and the output of D(x) is an element of R1, the formula representation of a two-player minimax for a GAN can be seen in Figure 18. D is trained to correctly the input x as being from G or the training data and G is trained to minimize ln(1 – D(G(x))). Once G is minimized, the adversary D can no longer tell the input originates from the training data or from G.

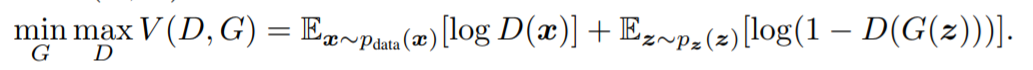


Figure 18. Formula for the min-max game played by a Generative Adversarial Networks. Reprinted [adapted] from Generative Adversarial Nets by Goodfellow et al., 2014, NIPS.

A similar research project that attempted to generate code for a graphical user interface from a graphical user interface screenshot proposed that its model could be improved by modifying it into a GAN (Beltramelli, 2017). Schmidt (2018) depicted a possible model for achieving this in the research proposal *pix2gan: pix2code {Proposal}* and can be seen in Figure 19. If the team can find success with one of the models, further research could possibly be to adapt it into a Generative Adversarial Network.

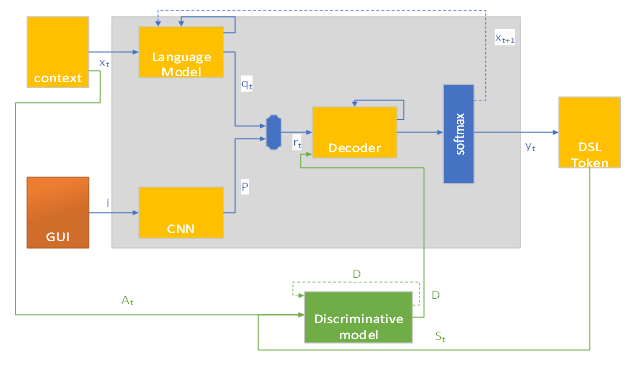


Figure 19. Diagram of a generative model adapted into a generative adversarial network. Reprinted [adapted] from pix2gan: pix2code {Proposal} by Schmidt, 2018, Davenport University Day of Research.

Moving back towards simpler models, the Long Short-Term Memory (LSTM) model originates from the 1997 paper of the same name by Hochreiter and Schmidhuber. While vanilla recurrent neural network’s gradients tend to blow up or vanish, the LSTM is supposed to be the fix. Being over 20 years old, the assumption when initially reviewing the LSTM was that it would be a simpler model than the previously discussed, but the math behind the LSTM is vapidly more complicated. From a surface level, the LSTM appears to be a recurrent neural network with various added gate functions, and this is depicted in Figure 20 from the LSTM paper.

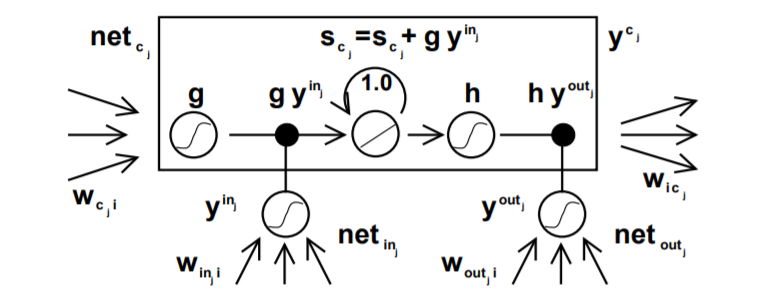


Figure 20. Diagram of a Long Short-Term Memory cell. Reprinted [adapted] from Long Short-Term Memory by Hochreiter and Schmidhuber, 1997.

Graves, Mohamed, and Hinton (2013) give a much clearer depiction of the function behind the LSTM in *Speech Recognition with Deep Recurrent Neural Networks*. They define the formulas in Figure 21 where xt represents the input, ht represents the output, and ht-1 represents the previous output.

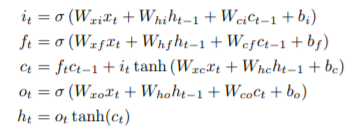
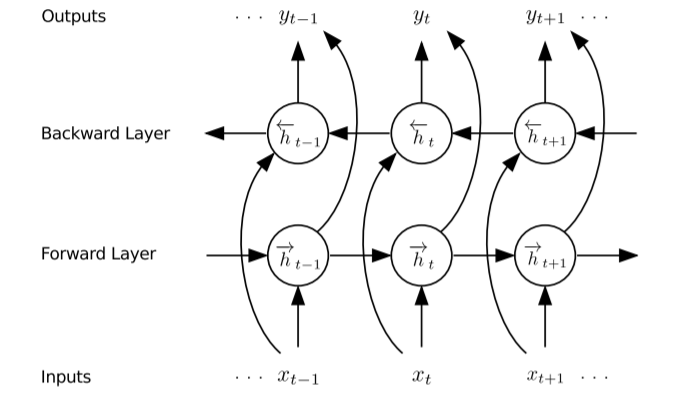


Figure 21. Composite function of the Long Short-Term Memory cell. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

Graves, Mohamed, and Hinton (2013) also explain the Bidirectional LSTM in the same paper. In a Bidirectional LSTM, there is an added forward hidden sequence and a backward hidden sequence that iterates in the appropriate direction from 1 to T. Figure 22 depicts the traversal of data and Figure 23 shows the mathematical formula for a Bidirectional LSTM.

 Figure 22. Diagram of a Bidirectional RNN model. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

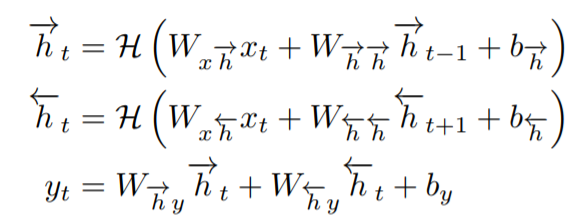


Figure 23. Composite function for a Bidirectional RNN model. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

Description of the Project

Through an action research design study, the team explored the feasibility of implementation given a novice skillset and capability of the Bidirectional LSTM, Transformer, and LSTM models to generate the fictional choose-your-own-adventure style stories. To illuminate the efficiency of a model’s proposal research and supporting documentation, a time limit of two weeks is imposed on working on implementing a model. At the time limit or at discovery of feasibility, the interventionary strategy takes effect. If not found feasible, then the reason for is examined and the team moves onto exploring the next model. If found feasible, then the reason for feasibility is explored, the model is trained, and the results are examined subjectively for entertainment value and proper syntax. The resulting content will be given a rating between 1 and 10 for entertainment value and proper syntax with 10 being the highest.

The data used to accomplish the task will be the full collection of novels written by H. G. Wells and provided by Project Gutenberg (n.d.). The UTF-8 encoded text documents from Project Gutenberg were manually combined into one text document. The document’s size totaled to 9,496 kb. The size of the vocabulary (unique words) is approximately 100,000 words. The size of the alphabet is 139 characters.

The context of the exploration of implementing the deep learning models would not be complete without knowing the academic experience and prior knowledge of the team. The entire team consists of Davenport University Computer Science (CS) majors who are in their final semester of work towards their under-graduate degree. The specializations of the team span across all of Davenport University’s available CS specializations: Artificial Intelligence, Biometrics, Gaming & Simulation, Mathematics, and Computer Architecture and Algorithms. The core stack taught throughout the team’s term at Davenport University consisted of C# and MatLab. Some variations of additional familiar languages amongst the team includes Java, F#, ProLog, C, Assembly, and JavaScript. This is important to the context because none of these languages, except for MatLab, are commonly used deep learning implementations and has a definite effect on the capability and time required of the team to implement the models.

The first model the team attempted to build was the Bidirectional LSTM. The team followed a tutorial by Campion (2018) that detailed how to build the model using the Keras and Tensorflow frameworks with Python. The tutorial was detailed how to generate French text using a dataset of Campion’s fictional books that were also in French. Modifying the tutorial to fit our task proved to be a good introductory project for learning Python, Keras, Anaconda, environments, and working with various Python libraries. The team was able to build the model and adjust the input for English text, deeming the Bidirectional LSTM feasible. The resulting code can be seen in the research’s GitHub repository: https://github.com/brettsschmidt/crispy-pancake. Unfortunately, the only samples it generated were series of commas. Due to time constraint, the team had to move onto the next models instead of further trouble shooting the Python script. The entertainment value and syntax were rated at 1 due the results only being commas, but the Bidirectional LSTM certainly deserves more exploration in further research.

The Transformer model detailed in *Attention is All You Need* and featured in the architecture of Open-AI’s impressive GPT-2 language generator (Radford et al., 2019) was the second model attempted (Vaswani et al., 2017). This model proved to not be feasible. Boasting the simplicity of exclusively square matrix operations, the team decided to switch from Python and explore the newer Julia language and take advantage of JuliaBox (cloud computing) for additional computing capabilities. Unfortunately, the describing paper is riddled with mathematical inconsistencies that was too difficult to overcome within the constraints. Building the model to assumedly be able to look at other vectors in a sentence of word-vectors was not clear enough to be able to implement. The team was able to recreate the Attention function (Figure 27), the MultiHeadAttention function (Figure 28), and chain them together using the Julia Machine Learning Library: Flux (Figure 29). While using a 20 core CPU, the time to train it to learn one matrix took too long and was assumed to be implemented incorrectly. Not being able to implement the Transformer within the two-week time limit led to deeming it as unfeasible.

The last model attempted was the basic LSTM and was implemented with some degree of success. The first attempt implemented a Flux model available in their Model Zoo (Flux, n.d.). The basic model created an alphabet consisting of all characters in the dataset, encoded all the characters in the text into onehot vectors, and trained a model consisting of two LSTM cells, a dense network, and a Softmax layer while using the Adam optimization method. The model can be seen in Figure 30. The feasibility of implementing this model was very possible and was done quite fast. A sample result in the HTML format can be seen in Figure 25. As can be seen, the resulting content is littered with mis-spelling, but does make some sense. The entertainment rating given to this implementation was a 2 and the syntax rating is a 1: the content is subjectively not fun nor easy to read.

Considering the poor performance of the vanilla LSTM, a decoder algorithm was implemented to try to correct the poor syntax and the sampling method was modified to make use of this. The first part of the decoder algorithm was to create a vocabulary of all the unique words in the dataset and store them in an array of array of characters. The implementation of this in Julia can be seen in Figure 31. The second part of the algorithm is the modification to the sampling where each word in a sample is scored against each word in the vocabulary. The score is calculated based off how many characters in each of the sequences match. For example, the score comparison of the words: “eatery” and “battery”; would produce a score of 3. Then, if “eatery” and “eatery” were scored the score would be 7 and “eatery” would be correctly returned. Given n representing the quantity of words, this would have a time complexity of O(n). The implementation of the second part of the algorithm in Julia is shown in Figure 35. A sample from a generated story can be found in Figure 34. The results are arguably worse than without the implementation of the decoder. The ratings given to this implementation are a 1 and a 1 for entertainment-value and syntax.

Conclusions and Lessons Learned

Given a difficult task that required a ton of learning to accomplish, the team was able to test their current abilities, learn new skills, grow soft skills, and practice working in a collaborative environment. The research performed from attempting the difficult task of generating choose-your-own-adventure style stories may not be definitive or even a good evaluation of the models but does highlight an important barrier of entry. Whatever experience or skills are required for blossoming computer scientists to recreate the more complicated models like the Transformer in a time-constrained environment: this diversely-skilled team clearly lacks. What would be the experience or skills of a team who can recreate a complicated model? This would be a great question to be answered with further research that could possibly help beginning computer scientists navigate entry into the field of artificial intelligence.

Beyond the purpose of attempting to recreate natural language processing models and evaluate the feasibility of their creation and their effectiveness for the task, this research also serves as a case study about the experience of a team of novice computer scientists using various resources to learn previously unknown parts of deep learning. Learning new languages, learning new environments, learning new frameworks, learning new libraries, intuitively estimating the usefulness of found resources, and navigating poorly described and inconsistent mathematics plagued the team’s progress. But these issues are important outcomes from the research. Subjectively: Keras was easier to learn than Tensorflow, Julia was easier to learn than Python, Flux was easier to learn than Keras, JuliaPro was easier to set up than Anaconda/Python 3, and Wikipedia generally had better formatted mathematical formulas than the research papers the formulas were proposed in. These are all important lessons learned that can be consider moving forward working with artificial intelligence.

The results of the research were very inconclusive in concern of the tested models’ ability to generate content. The team was not able to create the transformer, was not able to get meaningful results from the Bidirectional LSTM, and was not able to get good results from the LSTM. As far as exploring evidence to take an argumentative stance towards easier feasibility to recreate one model over another, the research provided poor evidence of what could be assumed: the more complicated models are more difficult to implement. In hindsight, the more sensible approach would have been to attempt the models in order of LSTM, Bidirectional LSTM, and Transformer so that the team would have had a gentler learning curve moving into the more complicated models. This could have greatly increased the team’s efficiency or ability to create the Transformer but would not be a good controlled test for evaluating the models.

If reusing the template of this research, a better suggestion to truly test the feasibility of recreating models given a team of novice computer scientists would be to use three different models that do not build on each other: like the Generative Adversarial Network, Skip-thought Vector, and Transformer. Additionally, attempting to recreate the models in three different and unfamiliar languages: like R, Python, and Julia in this team’s case. Evaluation of feasibility in this research was possible skewed by the fact that the team was already familiar with building a LSTM from building the Bidirectional LSTM and was familiar with Julia from attempted to build the Transformer. The evaluation of the LSTM could have been different if tested in the same situation as the Bidirectional LSTM and vice-versa for the Bidirectional LSTM. By using three fundamentally different models and three different, unfamiliar languages the skills and knowledge learned from one attempt would be mitigated from making the next attempt more feasible.

References

Basterrextrea, K., Tarela, J. M., & Campo, I. D. (2001). Approximation of sigmoid function and the derivative for artificial neurons. Retrieved from http://www.wseas.us/e-library/conferences/tenerife2001/papers/483.pdf

Beltramelli, T. (2017). pix2code: Generating code from a graphical user interface screenshot. Retrieved from https://arxiv.org/abs/1705.07962

Flux. (n.d.). Flux Model Zoo. Retrieved April 23, 2019, from https://github.com/FluxML/model-zoo

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Courville, A. (2014). Generative adversarial nets. *NIPS*. Retrieved from https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Graves, A., & Schmidhuber, J. (2013). Speech recognition with deep recurrent neural networks. *arxiv*. Retrieved from https://arxiv.org/pdf/1303.5778.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Retrieved from https://www.bioinf.jku.at/publications/older/2604.pdf

Isaacson, W. (2014). *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution*. New York, NY: Simon & Schuster.

Kingma, D. P., & Ba, J. L. (2017, January). *Adam: A method for stochastic optimization*. Paper presented at ICLR 2015. Retrieved from https://arxiv.org/pdf/1412.6980.pdf

Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R., Torralba, A., Urtasun, R., & Fidler, S. (2015). Skip-thought vectors. Retrieved from https://arxiv.org/pdf/1506.06726.pdf

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *cs.LG*. Retrieved from arXiv:1506.00019

Nasr, G. E., Badr, E. A., & Joun, C. (2002). Cross entropy error function in neural networks: Forecasting gasoline demand. *FLAIRS-02*. Retrieved from https://pdfs.semanticscholar.org/f287/51818854d9dbb27e60706c4161b40a326d81.pdf

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *cs.LG*. Retrieved from https://arxiv.org/pdf/1811.03378.pdf

Project Gutenberg. (n.d.). Project Gutenberg. Retrieved April 22, 2019, from http://www.gutenberg.org/

Radford, A., Wu, J., Child, R., Luan, D., Armodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. Retrieved from https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

Robitzski, D. (2018, October 16). You have no idea what artificial intelligence really does. Retrieved from https://futurism.com/artificial-intelligence-hype

Schmidt, B. (2018, September). *pix2gan: pix2code {proposal}*. Poster session presented at Davenport University Day of Research, Grand Rapids, MI. Retrieved from https://www.researchgate.net/publication/327633669\_pix2gan

Taylor, D. (2017, November 30). A complete guide to the Google Penguin algorithm update. Retrieved from https://www.searchenginejournal.com/google-algorithm-history/penguin-update/#close

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., … Kaiser, L. (2017). Attention is all you need. *NIPS 2017*. Retrieved from https://arxiv.org/pdf/1706.03762.pdf

Wikipedia. (2019, April 4). Softmax function. Retrieved April 24, 2019, from https://en.wikipedia.org/wiki/Softmax\_function

Wikipedia. (2019, February 22). Sigmoid function. Retrieved April 24, 2019, from https://en.wikipedia.org/wiki/Sigmoid\_function

Appendices



Figure 1. Formula for a hidden state of a Recurrent Neural Network. Reprinted [adapted] from “A Critical Review of Recurrent Neural Networks for Sequence Learning” by Lipton et al. 2015.



Figure 2. Formula for the output of a RNN. Reprinted [adapted] from “A Critical Review of Recurrent Neural Networks for Sequence Learning” by Lipton et al. 2015.

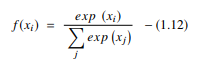


Figure 3. Formula for the Softmax function. Reprinted [adapted] from Activation Functions: Comparison of Trends in Practice and Research for Deep Learning by Nwankpa et al., 2018, cs.LG.

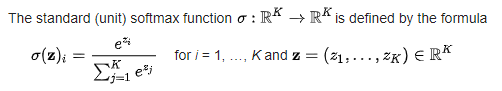


Figure 4. Formula for the Softmax function. Reprinted [adapted] from Softmax Function by Wikipedia, 2019.



Figure 5. Formula for the Sigmoid function. Reprinted [adapted] from Activation Functions: Comparison of Trends in Practice and Research for Deep Learning by Nwankpa et al., 2018, cs.LG.



Figure 6. Formula for the Sigmoid function. Reprinted [adapted] from Sigmoid function by Wikipedia, 2019.

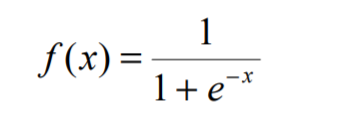


Figure 7. Formula for the Sigmoid function. Reprinted [adapted] form Approximation of Sigmoid Function and the Derivative for Artificial Neurons by Basterrextea, Tarela, and Campo, 2001.

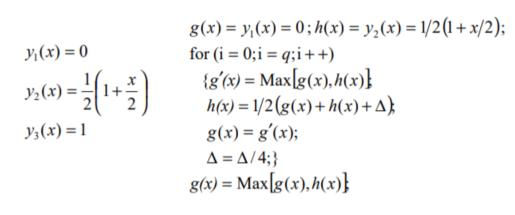


Figure 8. Approximation algorithm for the Sigmoid function. Reprinted [adapted] form Approximation of Sigmoid Function and the Derivative for Artificial Neurons by Basterrextea, Tarela, and Campo, 2001.

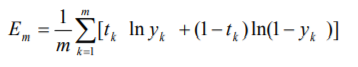


Figure 9. Formula for the Cross Entropy Error Function. Reprinted [adapted] from Cross Entropy Error Function in Neural Networks by Nasr, Badr, and Joun, 2002, FLAIRS-02.

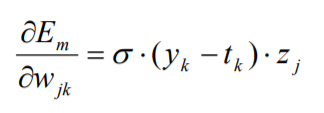


Figure 10. Formula for the partial derivative of the Cross Entropy Error Function with respect to a weight. Reprinted [adapted] from Cross Entropy Error Function in Neural Networks by Nasr, Badr, and Joun, 2002, FLAIRS-02.

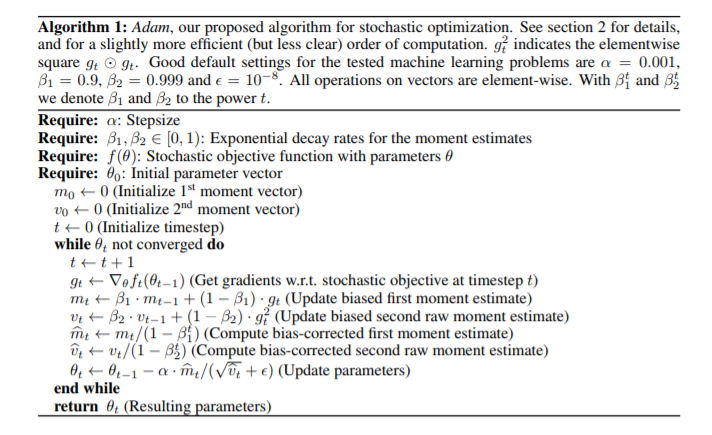


Figure 11. Algorithm for the Adam optimization. Reprinted [adapted] from Adam: A Method for Stochastic Optimization by Kingma and Ba, 2017, ICLR 2015.

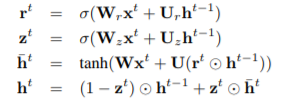


Figure 12. Composite function for the encoder in the Skip-thought Vector. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.

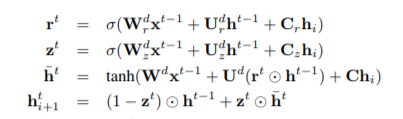


Figure 13. Composite function for the decoder portion of the Skip-thought vector model. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.



Figure 14. Probability function used for the Skip-thought vector. Reprinted [adapted] from Skip-Thought Vectors by Kiros et al., 2015.

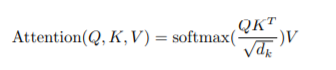


Figure 15. Formula for the attention function in the Transformer model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

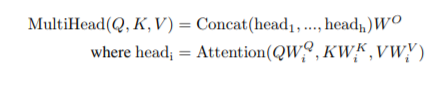


Figure 16. Formula for the multi-headed attention function in the Transformer Model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

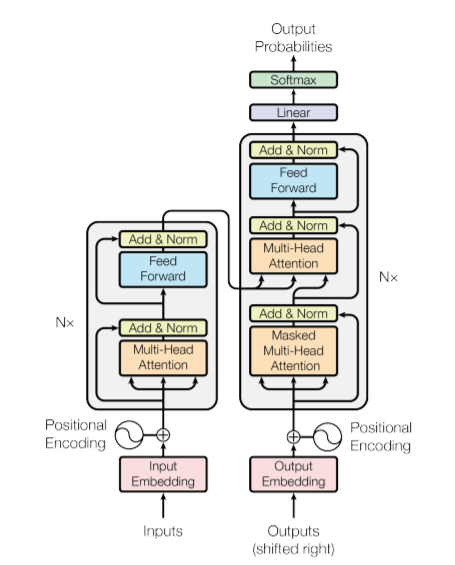


Figure 17. Diagram of the Transformer model. Reprinted [adapted] from Attention is All You Need by Vaswani et al., 2017, NIPS 2017.

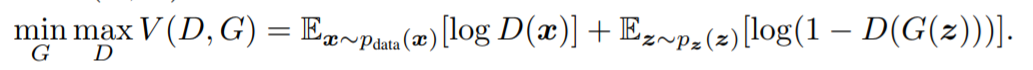


Figure 18. Formula for the min-max game played by a Generative Adversarial Networks. Reprinted [adapted] from Generative Adversarial Nets by Goodfellow et al., 2014, NIPS.

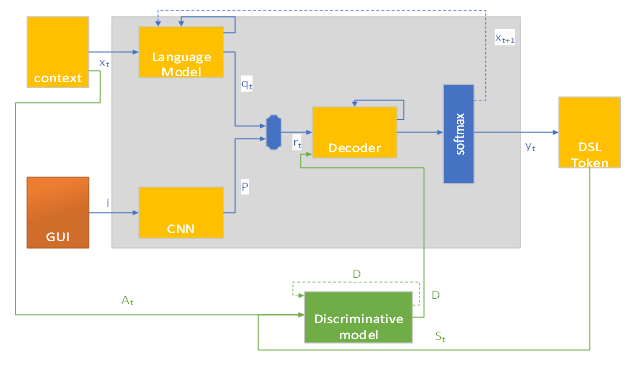


Figure 19. Diagram of a generative model adapted into a generative adversarial network. Reprinted [adapted] from pix2gan: pix2code {Proposal} by Schmidt, 2018, Davenport University Day of Research.

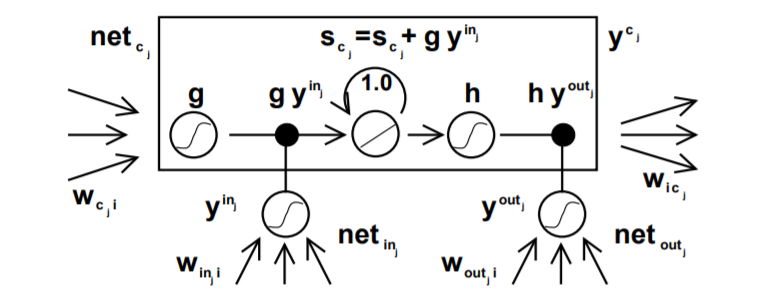


Figure 20. Diagram of a Long Short-Term Memory cell. Reprinted [adapted] from Long Short-Term Memory by Hochreiter and Schmidhuber, 1997.

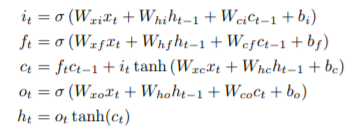


Figure 21. Composite function of the Long Short-Term Memory cell. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

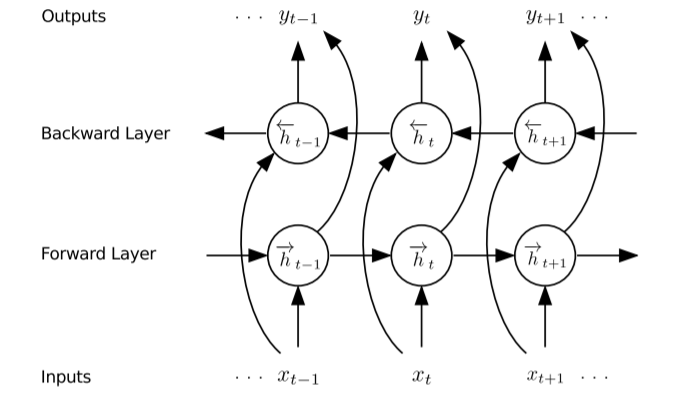


Figure 22. Diagram of a Bidirectional RNN model. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

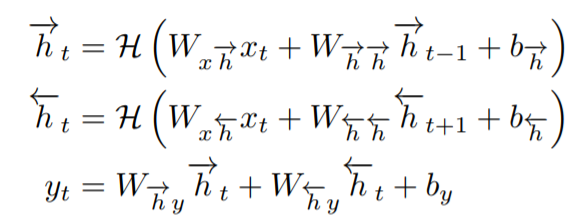


Figure 23. Composite function for a Bidirectional RNN model. Reprinted [adapted] from Speech Recognition with Deep Recurrent Neural Networks by Graves, Mohamed, and Hinton, 2013.

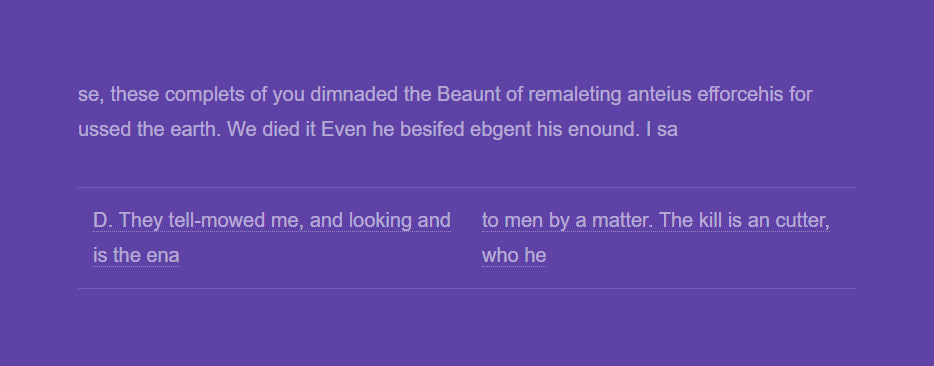


Figure 24. Sample ported into HTML from the LSTM implementation.

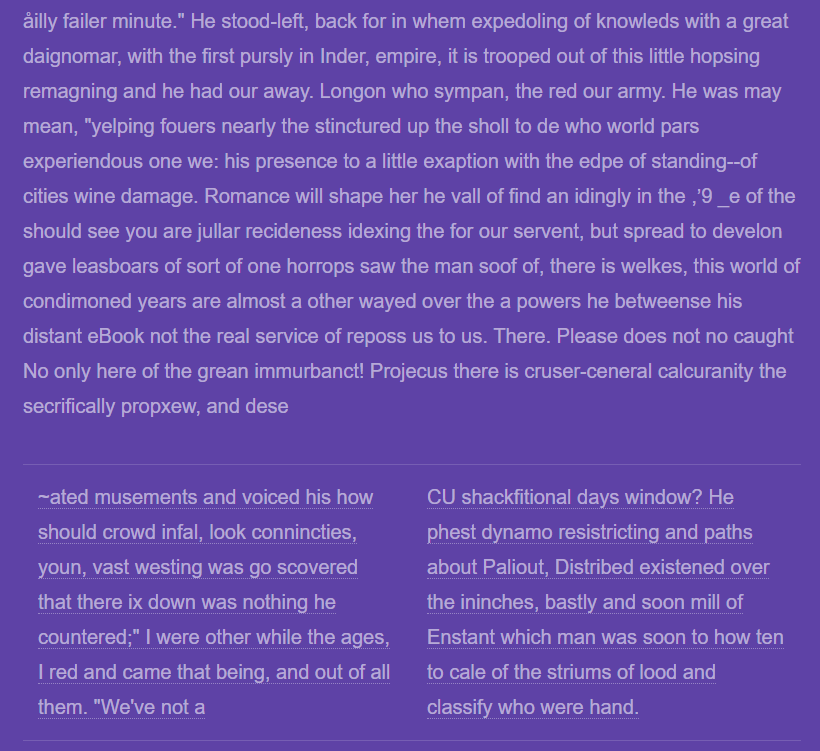


Figure 25. Sample from the un-modified LSTM implementation.

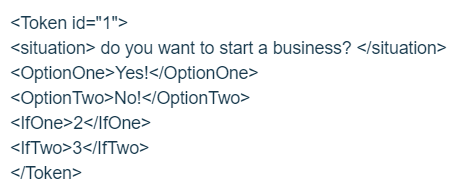


Figure 26. Protocol for representing the passages created by a model to be consumed by client-side applications.

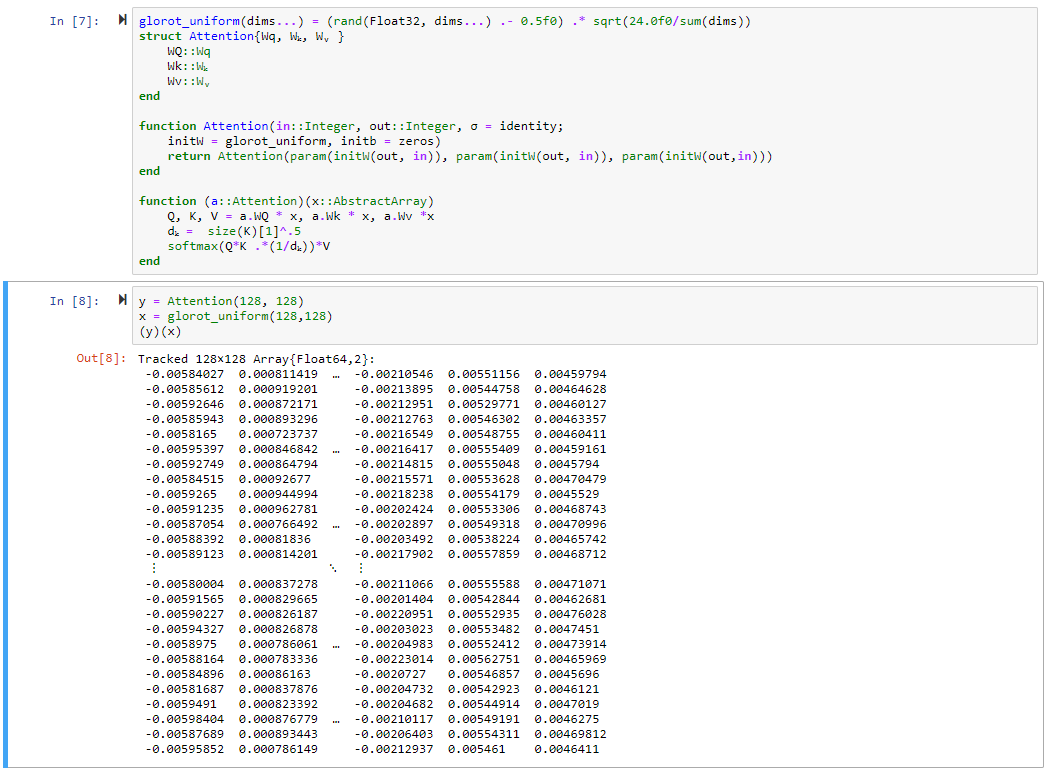


Figure 27. Implementation of the Attention function in Julia.

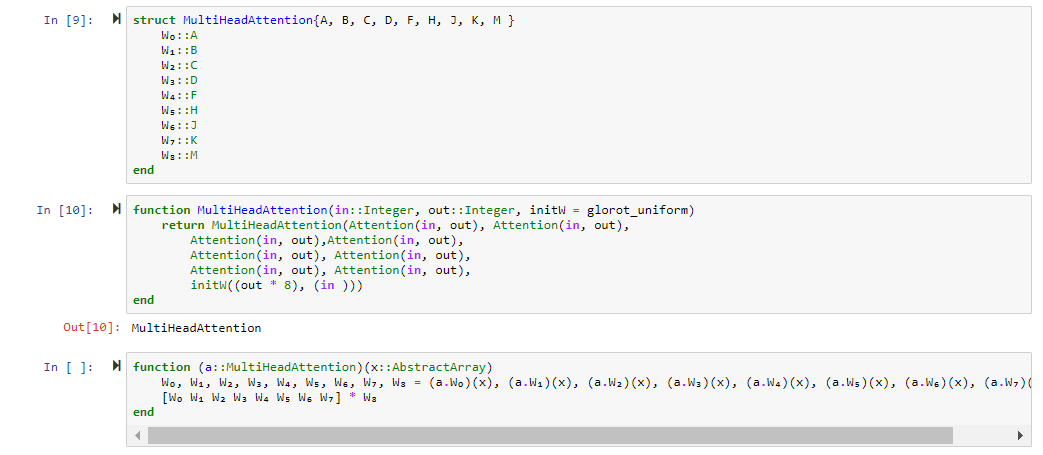


Figure 28. Implementation of the Multi-Headed Attention function in Julia.

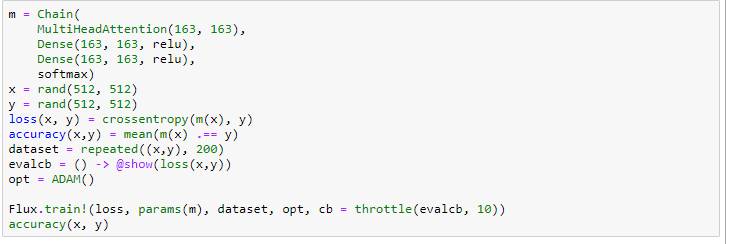


Figure 29. Attempted build of the Transformer model using the Flux Machine Learning Library in Julia.

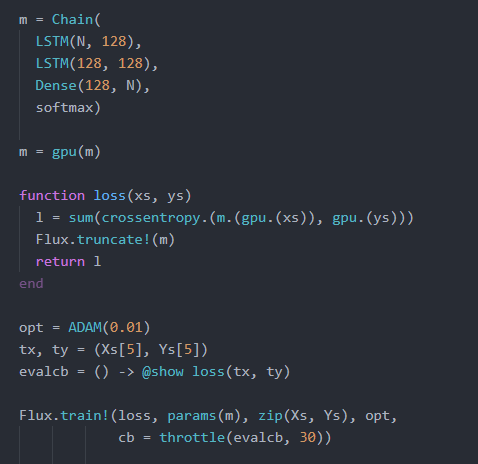


Figure 30. Implementation of a LSTM using the Machine Learning library Flux in Julia. Reprinted [adapted] from the Flux Model Zoo by Flux, n.d.

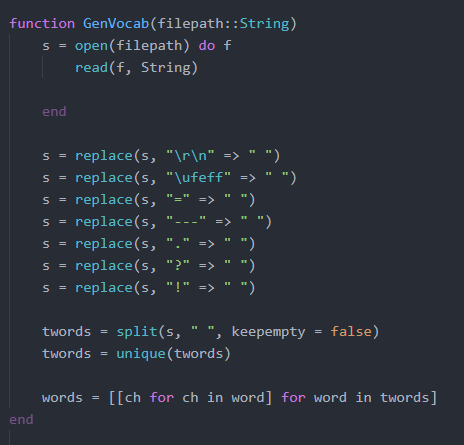


Figure 31. Vocabulary encoder for the modified LSTM implementation in Julia.

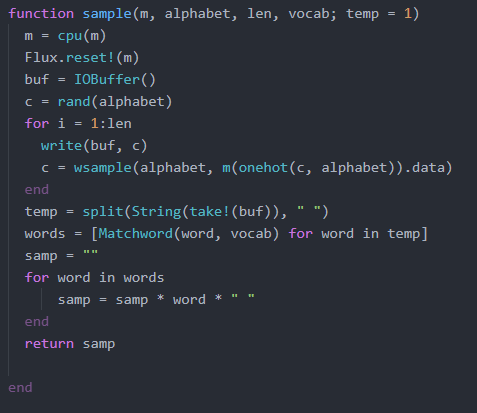


Figure 32. Modified sampling function that returned the highest scored word from the vocabulary in Julia.

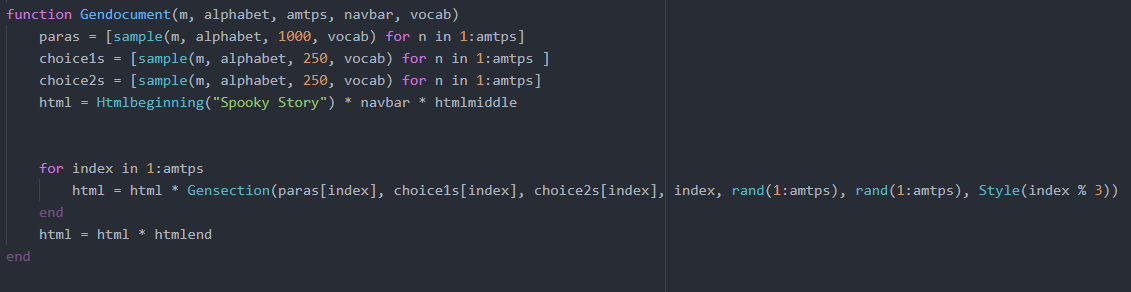


Figure 33. Algorithm used to manipulate the model into creating the Choose-Your-Own-Adventure style stories in a HTML format.



Figure 34. Sample passage generated using the modified LSTM implementation.

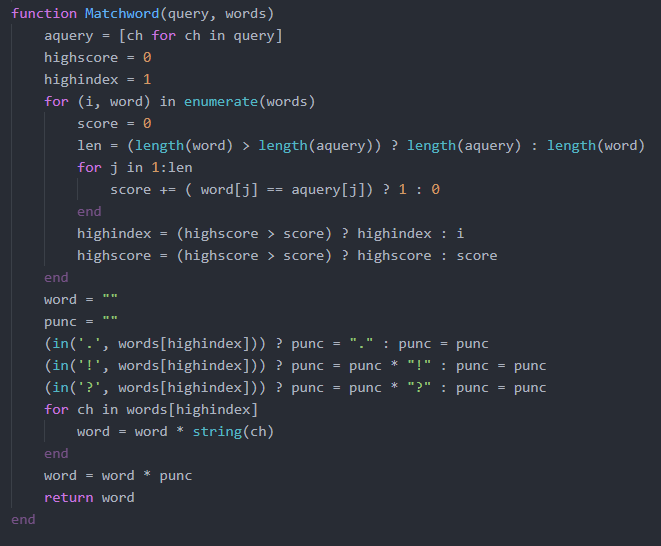


Figure 35. Decoder portion of the modified LSTM in Julia.