**Jonah Belback**

**Junior Design: Final Project**

**Text Generative ML through Reinforcement Learning**[**GitHub**](https://github.com/turtneck/ece1895-ML_JuniorDesign)

**Design Overview**

**High level description – Design**

I wanted to make a text generative AI based on datasets gathered by me or from examples I found around on the internet. This would be made with a neural network model using reinforcement learning, slowly becoming better at predicting, and therefore generating text. That, given a starting sentence, it could produce a paragraph of text that could be somewhat convincing or intelligible.

**Original Design concepts**

I originally started this project open ended, intending to switch the actual type of reinforcement learning AI if text generation was too difficult. I considered a video game like snake, or song/album names. However after finding out about the method of just predicting the next word, I just kept going in that direction. This method is much weaker, and doesn’t make great use of context of the previous words it’s building on, but it’s what I’m capable of right now.

In terms of “original design concept” I really didn’t have any besides wanting to try and make it text generative if I could, going in a different direction if I couldn’t. I’ve never actually done practical work in this field before, so I wanted to use this project to push myself to learn this topic. I do know a lot about the topic, however nothing strictly practical to the level of making it line by line.

**Expansion on previous work**

**Starting with Linear Relation**

I really enjoy the topic of AI, specifically reinforcement learning model. Whenever I get the chance to talk about a topic for a presentation it is my go-to. However I never actually made one before due to time or giving that time to my robotics club, so I just started out with a very basic model: a linear relationship between the temperature the previous day, to the temperature the next day.

I found these python courses through YouTube called ‘zero\_to\_gpt’ and just followed along. They already had datasets available, so it was less doing work and just learning the concepts.

Here is the data plot that temperature data, x=previous temperature, y=tomorrow’s  
A blue dotted diagram with numbers

Description automatically generated

A screenshot of a computer screen

Description automatically generated

This code is under dense.py in the ‘depreciated’ folder

[The github to the course’s follow-along examples](https://github.com/VikParuchuri/zero_to_gpt/tree/master)

[The video I followed with](https://www.youtube.com/watch?v=-cs5D91eBLE)

**Initial Text Generation**

I started my text generation model also based on a [course given on YouTube](https://www.youtube.com/watch?v=tEV_Jtmx2cc), however this code was just meant for a total walkthrough the concepts. It grouped unrelated text together for the training, it was all condensed on one file, the was no room to add more data to the set, and several other problems.

So using it as a jumping point, I created different libraries using what I learned and improved it and made it easier to train my model.

Additions:

* Separated files for different uses
* A total dictionary of words given to the model over the course of its history to use as indexing, allowing multiple datasets to be added
* Filtering of words
  + It was separating “don’t” to “don” and “t”
* Creating a new model file each time it’s trained with a different version marker

**Preliminary Design Verification**

This is the model right at the start just using the example code

A screenshot of a computer

Description automatically generated

The reason why it’s political was because the starting training data was fake vs real news headlines given in the example follow-along code.

After this was created, I split up work onto multiple files

1. Creating Model and Training the Model
   1. *WF\_create.py* which holds a specialized class holder for the Model
2. Creating Dataset from an input of text
   1. *data/csv\_seperator.py*, helped reformat and filter previous datasets
3. Running Model
   1. *WF\_train.py* which goes through all the datasets in given list of folders
   2. *WF\_train\_latest.py* which grabs the greatest generation Model and restarts its training at the next data set

Using the concepts and segmentizing, improving, and reworking sections from the example code to make them. After they were all made it was just grabbing more and more datasets, throwing it in to train the Model, and then reviewing how well the model is by running it after and manually evaluating it. If it seemed worse, I planned to run the current and previous version multiple times and decided if it genuinely decreased in quality, reverted back to the previous version. Although this never happened as it never got to a good enough quality to justify.

**Results per generation (1 to 10)**

All given same input  
Start text: *“He will have to look into this thing and he”*  
Number of words generated: *100*  
Creativity(what prediction take, 1=best): *1*

Note: The reason its political is because the dataset was taken from news articles

Gen1  
A black screen with white text

Description automatically generated

Gen2  
A screen shot of a computer screen

Description automatically generated

Gen3  
A screenshot of a computer screen

Description automatically generated

Gen4  
A screenshot of a computer screen

Description automatically generated

Gen5  
A screenshot of a computer screen

Description automatically generated

Gen6  
A screenshot of a computer screen

Description automatically generated

Gen7  
A screenshot of a computer screen

Description automatically generated

Gen8  
A screenshot of a computer screen

Description automatically generated

Gen9  
A screenshot of a computer screen

Description automatically generated

Gen10  
A screen shot of a computer

Description automatically generated

**Final Generations**

At this point I was worried about the success of this project and possible grading, so I downloaded everything to my Gaming PC Tower and had it run for as long as possible before the deadline.

Gen20  
A black background with white text

Description automatically generated  
“Trump to sue for accusations” is promising

Gen30  
A black background with white text

Description automatically generated  
The Sentences still don’t make any sense but were seeing more groups of one or two that do

**Design Implementation**

**Overview of full system**

I made a Machine Learning Text Generative Neural Network that evolved over multiple generations. It works by predicting the next best word at a time, keeping words in a dictionary by an index to make connections between them. By testing the model with more and more data, these connections will form stronger synapses between nodes with strong relations, and weaker ones with lesser relations.

You input the start of a sentence of paragraph, and the Model then predicts a number of words that it thinks would follow the other.

The Neural Network is sequential, starting at a Long Short-Term Memory (LSTM)\* shaped by its dictionary size, a second LSTM, a Dense layer that receives the input from all neurons of the previous, and then condensed to a single output in an Activation Layer.

\*An LSTM is a single unit (in this case there are multiple in this layer) that keeps track of values over a certain amount of time, forgetting unwanted information from the previous state. Keeping long-term dependencies while letting go of weaker inconsistent ones.

**Subcomponents**

Work was split up into multiple files focused on creating the best changes to the initial example code between them

1. Creating the Model
2. Creating and filtering datasets
3. Training the Model
4. Running the Model’s predictions

**Creating Datasets**

In the example code I followed-along to initially, it came with a giant dataset of 7796 both real and fake news articles, titles, and body text. However in the follow-along, he concatenated all of the body text into one continuous paragraph, capping it at 10,000 words (0.0335%) due to the size as without it and trying the whole dataset at once with 29,826,765 words requires 3.13 TB of allocated memory.

This makes a lot of the data lack context and makes the Model try to pair and learn from pairs of words from the end of one article to the start of another. This data also has lots of non-ASCII characters put in by mistake and these ended up in his model’s dictionary. It was also all in an excel sheet. So my plan was to train each of these articles individually, formatting and filtering all of it including extra spaces, newlines, and other unwanted characters that don’t fit in a dictionary.

So I made a helper file in my data folder called ‘csv\_seperator’ that took all the body text, cleaned it up, and dumped it into a text file. For algorithm reasons it’s all on one line.

A computer screen shot of text

Description automatically generated

This obviously filled up my folders, so I had to put it in its own. This folder is 27.4 MB

A screenshot of a computer

Description automatically generated

**Creating Model and Training Model**

I ended up making a new class object focused around a TensorFlow model with a bunch of helper functions in/outside of the class like:

1. Name tracking that can increase the version’s iteration or generation
   1. WB\_v0.0 to WB\_v0.1 (new iteration), WB\_v1.1 (new generation)  
      A screenshot of a computer program

      Description automatically generated
2. Colored printouts of training for visibility, rather than the regular printouts from the training that take too much of the terminal (adjustment allowed by model’s verbose flag)
   1. Later shown
3. Recursive training

More importantly is the function to train the model. Because the model has to have a number of nodes to the number of words it knows, something its created with, and they are rigged to specific indexes: if you add a new dataset with words the Model doesn’t know you have to remake the model’s nodes and retrain the model from scratch up to that point.

To best do this, I keep a dictionary file that the Model reads from. If the concatenated list of unique words from the dictionary and the dataset are equal, there’s no change in the dictionary and it can train no problem, this is an increase in its version iteration.

If it is not equal, it needs to be remade with the new dictionary which file’s is then updated. This is a change in generation. Every time the model is trained it has kept a history of what dataset it was trained in order. This includes duplicates. Because the training of datasets are deterministic, not up to Chaos Theory, you can recreate a Model as long as you train it with the same datasets in the same order, so we recreate the Model up to the point of its previous generation but now with a number of nodes indexed to a new dictionary. Then you can train the brand-new data set.

Assuming that the Model is continuously introduced to new words in each data set, with n = number of generations, it has a runtime complexity of θ (n!), while any iterations are still θ(1). This is very wasteful in computational time, but I currently do not know a better method.

In fact it takes so long that I can’t even get through my whole news data set. It would take too long to make this report.

I decided just for this report, I am limiting the total amount of datasets the model trains with is 10. This still takes 30minutes. However you can already see that as it reached the end of the 10th article, the Model was getting an accuracy loss of nearly zero. In the giant concatenated version of the articles it was 0.3 to 0.9 with 10,000/29,826,765 words or times more data.

A screen shot of a computer

Description automatically generated  
My method (10 articles, 6720 words)

A screen shot of a computer

Description automatically generated  
The example’s giant concatenation (limited to 6720 words)

So given the same data, the same words, the same articles, the initial example with giant concatenation, while waaaaay faster, is much less accurate, even towards the end of the training its accuracy only ever reached 0.9154. While my method of individually testing each article as a new dataset, recreating itself every time the dictionary changes, has an accuracy loss of near zero towards the end of its’ finally dataset and the end of the 6720 words.

There is also much more room to train the data my way as it works off data that is already stored on the computer rather than allocating the entire suite of datasets at once in python.

I’d say that’s a successful improvement.

If I had to propose a solution, it would be to add up all the words of the all the datasets you have and want to use on it at its initialization and create a super dictionary to make the nodes for. Then it can only create iterations, not generations and each dataset is a complexity of θ(1). A massive improvement. However as of the state of the codebase with the time I have left, I cannot restructure my entire architecture in time for this report. I am actually doing paid ML research this summer for SHREC, so I might just improve this on my own time after this semester on the side.

This would also likely run into the same problem as concatenation where you allocate too much space in python at once, and my laptop is not equipped with a TB of RAM.

Maybe you could start with a preexisting super dictionary, and any word read in a dataset not included in the dictionary is just ignored. I would be willing to try that however, I really want to see how far the original setup can go up to 2 hours before I have to submit this report.

**Design Practices**

The design practices I put in were trying to segment the work involved and making the flow of work simpler to go through.

Also making my datasets clean with filtering and refracted to increase the gains from training as discussed earlier.

Creating my own printouts and muting the default (seen in the word comparison the section before) made readability of watching the terminal during training way more comprehensible. Especially with how many times the Model has to redo previous training, you had to really scroll and often text was cut off by the line limit in VS Code’s terminal.

A screenshot of a computer program

Description automatically generated

I feel like I can actually breathe looking at this. It also helps to when training takes literal hours to have the time to do this.

I also made a separate testing file that starts at the most evolved model and picks up it’s training from there. There are also shortcuts in the file that runs the Model’s predictions that skip user inputs to just use default values when you just enter an empty value (brushing by with the enter key). You can also have it automatically go through every model with the same inputs in the prediction function to compare the different versions of the models. This allows me to have failsafe’s and get through testing faster and more comprehensively.

**Design Testing**

**Test Plan**

After making all the infrastructure code, it was all just training the model which went like this:

1. Grabbing datasets
2. Reviewing the output before training it
3. Training the Model to a new generation
4. Running the new Model several times with new predictions
   1. If it was worse than the previous model, scrapping it and skipping over that dataset
      1. This never happened

Also the additions mentioned in the last section to aid in testing and training.

**Assembled Prototype**

This is the final generation of the Model: version 35

Input start text: Please help me and convince Dr.Dickerson to give me a good grade  
Words generated: 100  
Creativity(# place option 1st to last): 1 (best)

“““

Please help me and convince Dr.Dickerson to give me a good grade imprison floor more hampshirestill fdic brief repeatedly reddit bulldozer third markets onions matthew didnt university could ho on into acknowledged allowed however his was a wasnt better for narrative warming tore are ibd this up well disaster people liar that tore not like then same blanchette climate at like people and there hurricanes destruction poliird markets onions matthew didnt university could ho on into acknowledged allowed however his was a wasnt better for narrative warming tore are ibd this up well disaster people litical who 0 than was president and a the was am less couple of which up as for to to coast disappointed death human outside and and that as dhe was am less couple of which up as for to to coast disappointed death human outside and and that as damage weatherrelated he for global the global and for same warming the if eaamage weatherrelated he for global the global and for same warming the if early and earth that

” ” ”

A black screen with white text

Description automatically generated

**Functionality state**

I planned on finding a bunch of more datasets, but after the spiral of improving how the model was trained with data compared to the initial example, this was not possible. Even if I let it train for the entire day, it wouldn’t reach the end. If I had reached this state weeks ago then maybe, but its lack of accuracy loss in training makes me feel not that bad about it. Training just takes a very long time, but I built the infrastructure necessary.

**Final Presentation**

<https://www.youtube.com/watch?v=lTBftMP1bzY>

**Summary**

I learned more in this project than I have in any others I’ve done for any of my classes. It’s a topic I’ve been interested in and wanting to tackle for a very long time, and I had a lot of fun doing.

Obviously, my model still needs a lot of work, as the latest generation is still intelligible. Machine Learning is specialized for finding underlying mathematical relationships, if given data on a graph, it’s going to find the formula for a line of super-fit. But in this case, it’s trying to find relationships between seemingly random numbers that only serve as indexes for words it can’t have direct access to. In order to overcome this massive task of making it do something it wasn’t directly made for requires an incredible amount of data and training which was not something that can be done to the level needed for Junior Design. I plan on either seeing it through on my own time or though one of the research groups I’m a part of during the summer.

I am really excited that I was able to do this and drastically improve the ways the model was trained with my own ideas.

Runtime of training per number of successively dictionary changes:

Ran uninterrupted with minimal background processes on dedicated gaming PC

I was wondering if, as more datasets and words are added, less and less unique words would appear, and the runtime would curb. But the factorial complexity seems to win over.

If I had to change anything about this project was purely just doing more work earlier, not knowing how much work opened up when I thought it was closing out.

I also would have made all my dataset myself rather than having to try to reformat and filter someone else. I realized this much too late.