CSC 7700 Foundational AI Project 2 Report

Samuel Hildebrand
shilde2@lsu.edu

Abstract

In this project I programmed and trained four small-scale language models using historical architectures up to and including transformers. The architectures used were Gated Recurrent Unit (GRU), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Transformer. All models were trained and evaluated using a dataset derived from the Project Gutenberg online library. Of all models trained and evaluated, unsurprisingly the model which utilized the transformer architecture preformed best, with a Perplexity (PPL) of 100.89, and a BLEU score of 0.0120. PPL and BLEU scores for all models are given in the results section of this report.

Methodology

Initially, I finished the implementation of the Gated Recurrent Unit (GRU), provided by Dr. Ghawaly. Since I finished the implementation, I've included results and a training graph of GRU in the results section, as well as further implementation details for GRU in the methodology section as a point of comparison for my other models. Once the Gated Recurrent Unit model was able to be trained successfully, I wrote the files `generate_text.py` and `evaluate_model.py` to generate text given a prompt, and calculate Perplexity (PPL) and BLEU scores for my models. Detailed usage information for both of these scripts can be found in the README file of the included GitHub repository.

With GRU implemented, I moved on to Recurrent Neural Network (RNN) implementation. I began implementing RNN by copying the python module I had created for GRU, which contained a file detailing the `GRULanguageModel` object, and a script to train the model. As I neared the end of implementing RNN, I began to realize I was rather egregiously violating the DRY (Don't Repeat Yourself) software development principle. To remedy this, I created the project organization structure as it now exists:

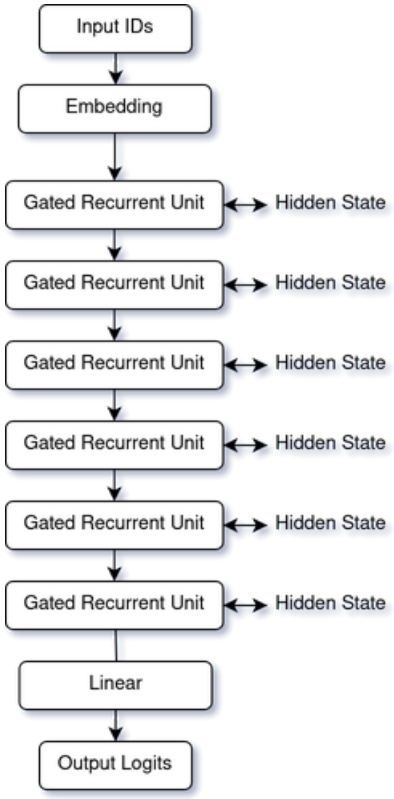
- The **root** project directory contains scripts for training and interacting with models, as well as this abstract and a README for script usage information.
  - There are 4 python modules for the language models
    - **GRU**
    - **RNN**
    - **LSTM**
    - **Transformer**
  - There is a **BaseLanguageModel** template module that all of the language models inherit from
  - There are 2 helper python modules that all the language models call
    - **TextDataset** which creates a PyTorch Dataset that the models can use from our dataset
    - **Tokenizer** which handles formatting the dataset and training the tokenizer. There is a script in **Tokenizer** which must be run to train the tokenizer before training any models. Again, documentation for all scripts is in `README.md`

With the project organized in this way, each language model module only needs to specify an `__init__` and `forward` function, since all the other functions do not change from model to model and are inherited from BaseLanguageModel.

Each model allows the user to specify a sampling temperature as a float, and in my implementation, if 0 is specified as the sampling temperature, greedy decoding is used.

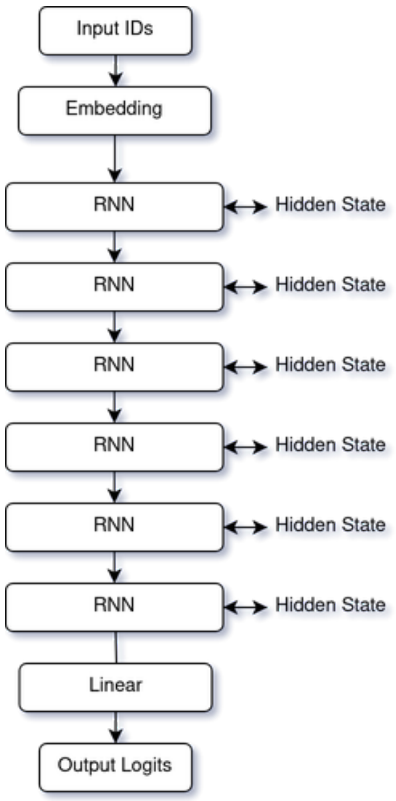
Gated Recurrent Unit (GRU) Architecture

I finished the implementation of Gated Recurrent Unit (GRU) provided by Dr. Ghawaly in lecture 6. I continued to utilize a 6 layer `torch.nn.GRU` in that implementation. I would like to revisit this project at some point and explore several different layer numbers for each model, but due to time constants, especially considering training times, I used 6 layers in each of my models. An architecture diagram of my implementation of a Gated Recurrent Unit (GRU) model is included below:



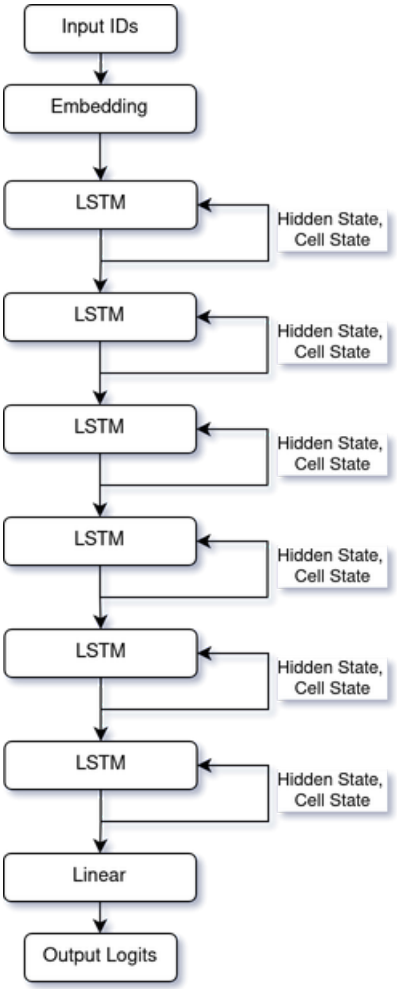
Recurrent Neural Network (RNN) Architecture

Like with GRU, I used 6 RNN layers in my implementation of a Recurrent Neural Network model. An architecture diagram of my implementation of a Gated Recurrent Unit (GRU) model is included below:



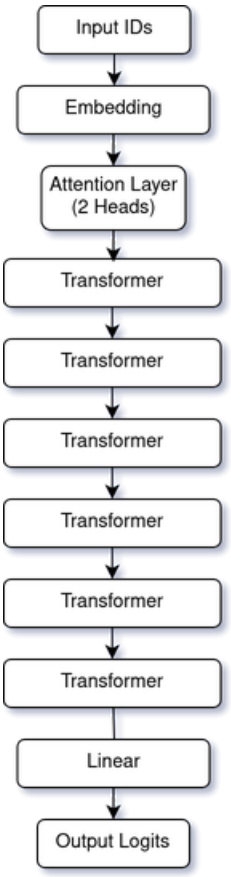
Long Short-Term Memory (LSTM) Architecture

As mentioned above, I think the 6 layer architecture that I used for my LSTM implementation was not ideal. However, an architecture diagram of my implementation of a Long Short-Term Memory model is included below:



Transformer Architecture

Again with my implementation of a Transformer language model, I used 6 `torch.nn.TransformerEncoder` layers, as well as a `torch.nn.TransformerEncoderLayer` with 2 attention heads. An architecture diagram of my implementation of a Transformer model is included below:



Results

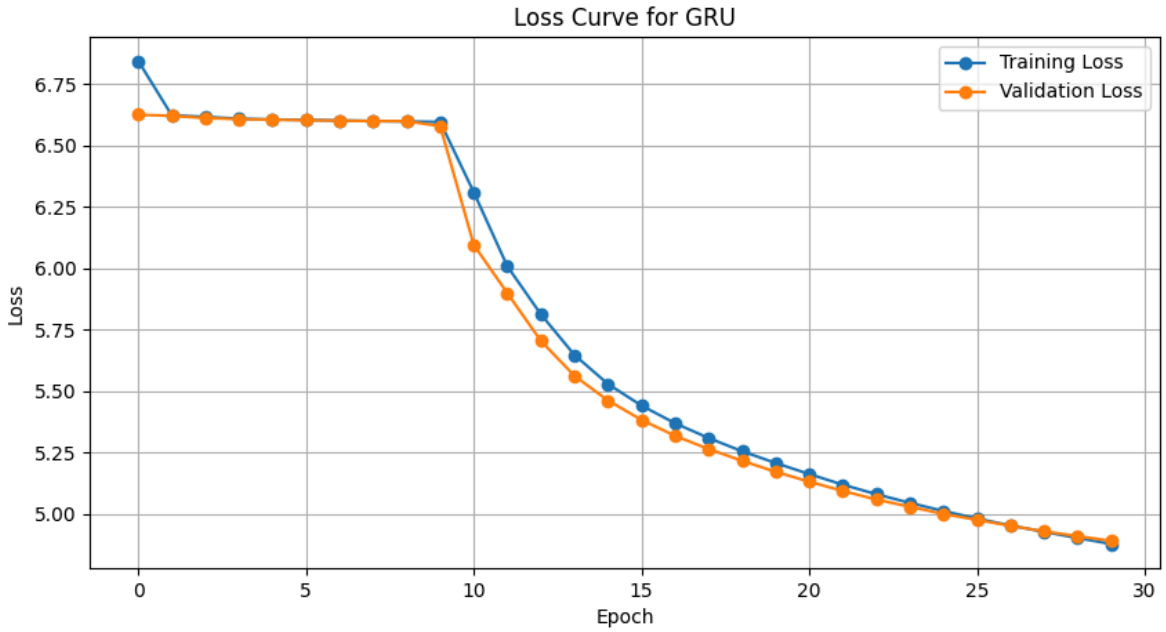
I have included a table of the Perplexity (PPL) and BLEU scores for all the models I trained and evaluated below. Overall, I found the Transformer model to preform the best, in the evaluation metrics as well as intuitively generating more coherent text. GRU outperformed RNN, which seems unsurprising, but LSTM preformed worse overall, which I found a little surprising. Perhaps my 6 layer LSTM architecture was unsuitable and more layers would have been preferable.

Model	Perplexity (PPL)	BLEU Score
RNN	136.11	0.0098
GRU	133.10	0.0100
LSTM	182.06	0.0082
Transformer	100.89	0.0120

For Comparison: Gated Recurrent Unit (GRU) Results

I have included the results from the Gated Recurrent Unit model since I did implement and train it, and I found it to be somewhat helpful when comparing results from some other models, especially RNN.

Included below is the loss and validation curve generated while training the Gated Recurrent Unit (GRU) model:



For each model, I've generated a maximum of 50 tokens given the prompt *"Which do you prefer? Dogs or cats?* at a sampling temperature of `1.0`. Included below are the results for Gated Recurrent Unit (GRU):

```
python3 generate_text.py --model_path GRU "Which do you prefer? Dogs or cats?" --max_length
50 --temperature=1
```

```
Generated Text:
said,, the least in the debtate him. ⁇ sally from the pillow, hes to mes you? ⁇ , the
heavens figure, I said my soul, you so master, Nicholas is, though he had swept should go to
```

The `⁇` indicates an unknown character. I'm not sure what in the training dataset is being sampled that shows as an unknown character. It is also possible that this is a bug in my tokenizer implementation, however this unknown character shows up predominantly in text generated by the GRU model.

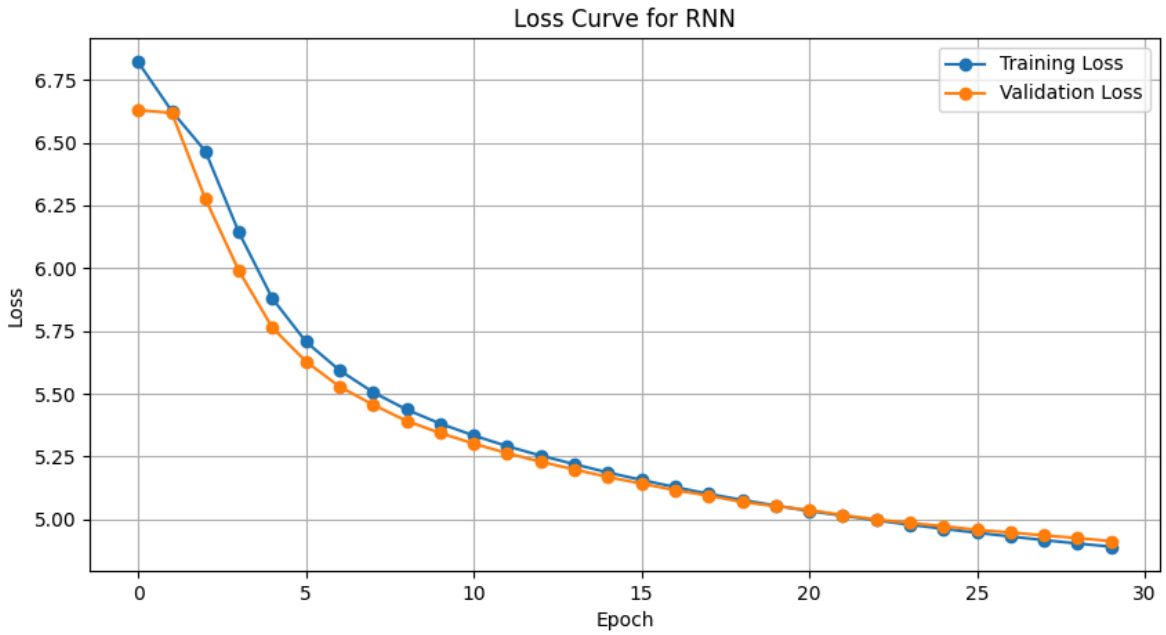
For my next prompt for each model, I choose to ask a simple question any supercomputer should know: "What is the meaning of life?". With a temperature of 0.5 to show off specifying temperature for sampling, I noticed that many of the models waxed weirdly philosophical, in a nonsense sort of way. I think these models and some generated scenery could farm a lot of likes on certain parts of social media.

```
python3 generate_text.py --model_path GRU "What is the meaning of life?" --max_length 50 --
temperature=0.5
```

```
Generated Text:
When he was so, it, the other in the other, the house, it. ⁇ s of the first, I have happened
to the merchantd of the cylinders, I was not been made the same, I have been of the in
```

Recurrent Neural Network (RNN) Results

Included below is the loss and validation curve generated while training the Recurrent Neural Network (RNN) model:



For each model, I've generated a maximum of 50 tokens given the prompt *"Which do you prefer? Dogs or cats?* at a sampling temperature of `1.0`. Included below are the results for Recurrent Neural Network (RNN):

```
python3 generate_text.py --model_path RNN "Which do you prefer? Dogs or cats?" --max_length 50 --temperature=1
```

```
Generated Text:
Oh, than a chaff petans prompt to put out possession from business mes waters always bore out and laid on the high it is out under the greatest splendor of substancesggmer characteristic of the girl discovered! Certainlyston surgeon to range man
```

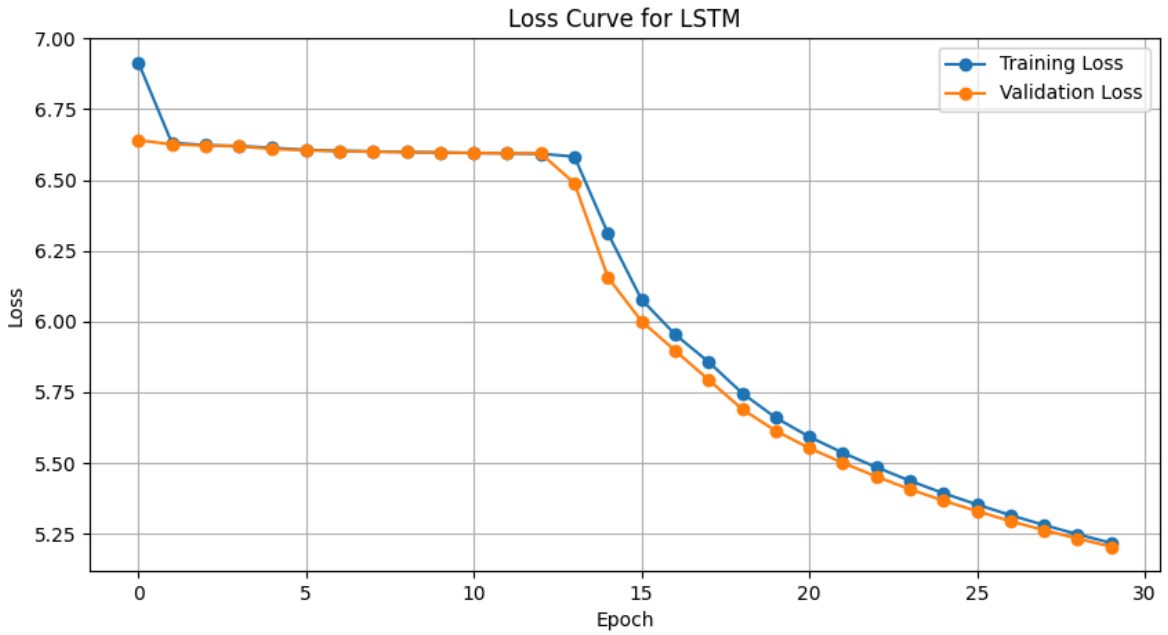
For the next test, I generated a maximum of 50 tokens given the prompt *"What is the meaning of life?"* at a sampling temperature of `0.5`. Included below are the results for Recurrent Neural Network (RNN):

```
python3 generate_text.py --model_path RNN "What is the meaning of life?" --max_length 50 --temperature=0.5
```

```
Generated Text:
I shall have been in the poor. Well. And the doctor. My birth. It was the same, according to the princess, assuming to the same, when he would be a hundred minutes in the same, among the body of the same,
```

Long Short-Term Memory (LSTM) Results

Included below is the loss and validation curve generated while training the Long Short-Term Memory (LSTM) model:



For each model, I've generated a maximum of 50 tokens given the prompt *"Which do you prefer? Dogs or cats?* at a sampling temperature of `1.0`. Included below are the results for Long Short-Term Memory (LSTM):

```
python3 generate_text.py --model_path LSTM "Which do you prefer? Dogs or cats?" --max_length 50 --temperature=1
```

```
Generated Text:
As the father, ascended to work fresh and throwing go for Barclays nor the gentleer. the firstad having burned himself Alcinous inspirecia upon lets. Here to comesI go for impent of board to be of any hesitating vengeful
```

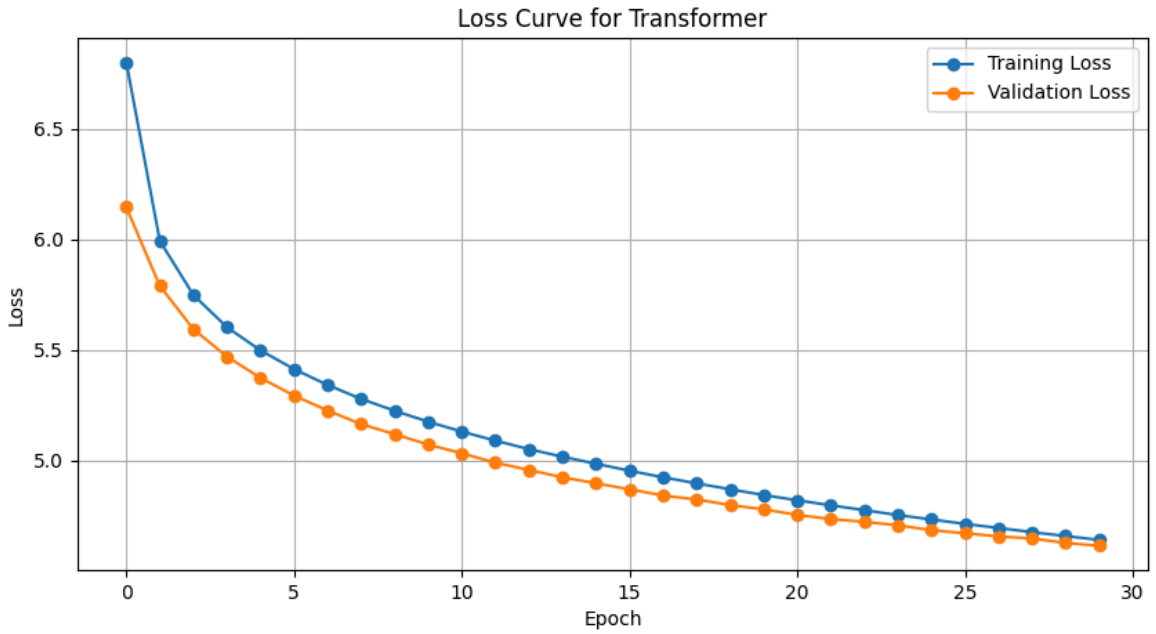
For the next test, I generated a maximum of 50 tokens given the prompt *"What is the meaning of life?"* at a sampling temperature of `0.5`. Included below are the results for Long Short-Term Memory (LSTM):

```
python3 generate_text.py --model_path LSTM "What is the meaning of life?" --max_length 50 --temperature=0.5
```

```
Generated Text:
A. I may be, I will be going, you wakes, the low, he was the rest, the water, they have been there is been, the little father, he was, I had be., that the door, the
```

Transformer Results

Included below is the loss and validation curve generated while training the Transformer model:



For each model, I've generated a maximum of 50 tokens given the prompt *"Which do you prefer? Dogs or cats?* at a sampling temperature of `1.0`. Included below are the results for the transformer model:

```
python3 generate_text.py --model_path Transformer "Which do you prefer? Dogs or cats?" --
max_length 50 --temperature=1
```

```
Generated Text:
And try to hurry, in the genius against you are sold his legs to proceed from the
alternative of excellency. The steward, and of trees, misery and Miss Bourgh with olimored
glass to say. I speak, be ready to place the
```

For the next test, I generated a maximum of 50 tokens given the prompt *"What is the meaning of life?"* at a sampling temperature of `0.5`. Included below are the results for the transformer model:

```
python3 generate_text.py --model_path Transformer "What is the meaning of life?" --
max_length 50 --temperature=0.5
```

```
Generated Text:
I saw the other, and that he found the gods, that the same way, and though you have a man,
and the same. The first and then, and his eyes, to be a man, stood out of the counted me,
```

Code Repository Link

https://github.com/Sam-Hildebrand/Mediocre-Language-Model-at-Best

Conclusion

Obviously, these language models do not compare very favorably to modern LLMs like Llama, ChatGPT, Deepseek, etc. And, except for the idea I mention earlier about the meaning of life responses, these models seem to really only be useful for providing a good laugh. As stated in the results section, the Transformer model preformed best and the LSTM model preformed surprisingly poorly, although my architecture may be to blame for that.

Other than the obvious exposure this project has given me to older language model architectures (and transformers, which will be very useful to me) I'd say the most valuable thing I gained from this project was the exposure to PyTorch, which I didn't have a lot of before this class and this project specifically. I tried to preform the necessary checks so that my models will train and run on CUDA, MPS, and CPU. I've also started to become more conscious of my coding practices since starting graduate school, and projects like these force me to consider good object oriented style in this case, and good coding practices like DRY. This was a fun project, and I was very amused by some of the responses from the models.