1.)

Hi, my name is Jonah Belback, and my project was on Prompted Response Language Learning Models.

I really wanted to learn how things like Chat-GPT were able to not just generate text but actually answer questions. Last semester I had worked on a very basic LSTM model, but all it could do was continue some input text with some half-connected garble.

This was really challenging because I’ve yet to take any classes in Machine Learning, I’m doing that next semester. And when I would learn something and try it out, I wouldn’t know what I did wrong or right until training was done a week later.

So my approach over the summer was just to take it one step at a time, research some section of LLMs, try it out by hand, experiment with a bunch of different approaches that I thought up, let it run, and then come back and see what worked and didn’t.

I did this because there were a lot of things I would learn about, be told the correct approach to using it, and not explained why exactly this other thing couldn’t work. Or sometimes even what WAS the best thing. And I wanted to know why and find out the hard way.

So I have 6 different quote on quote “finished” models to show off today, with all the training I could fit in for the expo.

~~Here’s a loss graph of all the different trainings of them put together, and what I think is the best response I found between all of them.~~

Heres my best looking loss graph from all my trainings and what I think is the best response to a prompted question was that I found between all of my models.

I honestly don’t have much regrets, obviously everyone wishes they spent more time doing something, but I think I did the best I could with the time gatedness that training takes.

Theres more that I can learn on topic, and I plan on continuing this project past this expo, to learn more about those things like making a full transformer which is something I failed at.

Because my work over the summer wasn’t one set project, I’m going to be using a lot of supplemental slides so let’s get into that.

2/3.)

So the first thing I tackled was data, my biggest regret making my previous LLM model was the lack of good data. So I found Gutenburg.org the largest digital public domain library. I just made a quick data-scrapper that wasn’t super aggressive on server requests and that gave me over 57 thousand books.

4.)

Later on in the project I need data with prompts and responses, and I found that on HuggingFace.co which is an public ML community.

I got a lot of help from Open-Orca with their datasets, and some more natural chatbot responses from Movie Corpus.

5/6.)

A major part of LLMs is making text actually understandable for Machine Learning. Models predict things, give it X, and it will find Y.

So getting it to predict text means breaking text down unique sections assigned to number, or split into tokens, is tokenization.

7.)

Theres three types of tokenization, By-word, By-Character, and by Subword.

This was my first experiment, making my own dictionary for By-Word and Character, and how well it a model can actually use that. But was short lived when after going through the entire Gutenburg library, By-word was well over 7 million tokens. So I went on to compare By-Char and By-Subword using tiktoken which you’ll see at the end.

8.)

One experiment I did while waiting on training was comparing how fast datastructures could put together a dictionary for a tokenizer. I thought maybe characteristics like sorting by frequency of each token would help so there’s less comparisons to identify if a token is unique before adding it, but unbridled RAM access was the definite factor. Something like Unsorted list, comparing every single token before adding, still blew out everything in terms of time.

9/10.)

Getting into models, one of the models I have prepared today is a’ Sequence to Sequence’ Model.

It is the predecessor to the LSTM, the original model I made.

The main difference between Seq-to-Seq and LSTMs is that, while both being RNNs, LSTMs address a really important problem which is something called “Vanishing Gradient” which is an error dealing with learning with very little loss. Basically it can’t pick up on and learn smaller details.

And the difference between Seq-to-Seq or LSTMs and Transformers is that Transformers pay attention to data simultaneously rather than sequentially.

11.)

Transformers are made up of an Encoder and a Decoder like Seq-to-Seq and LSTMs. And both parts still serve the same purpose:

The encode learns and understands the text, while the decoder more so generates.

But when it comes to training Transformers, the Decoder is done first and completely separate with something called pretraining

12.)

This is where the decoder is trained off raw text and finding the next token in sequence after looking at the all the tokens it given to look at, or its context. Which is the max number of tokens the Model can look at and process to determine an output.

When this Decoder goes to generate, it would do so one token at a time looking at the whole context its given, adding a token that’s its best guess, and then reentering the with it appended to the initial context to find the next token. This is what the Gutenburg dataset is for.

Chat-GPT2 is actually just this decoder module.

I have 4 different models prepared; split between 2 of 2 types of variants I’ll get into later.

13.)

This is what a whole transformer looks like, and is basically the defacto standard for LLMs with maybe some additional feeds into LSTMs. It can be used not just for LLMs but for image processing, audio generation, and even Chess.

This was not something I was able to create in time. There were things I wanted to try out and learn from just the decoder Model which is why I’m showing 4 different ones of that. This was actually my next step and what I’m working on past this expo before next semester.

14.)

The reason I stuck around with the Decoder Model instead of just jumping to the transformer was that after making the Seq-Seq in-between I wondered if you could use its technique to train with prompts to work for the Decoder.

The actual architecture for an encoder and decoder transformer are almost identical, and a decoder couldn’t generate continued text if it couldn’t identify meaning from it. So if a Seq-Seq picked apart info from a special identifier separating the Question and Answer, why couldn’t just a decoder?

If a decoder could pick up on this dividing SOS-token, could it generate a response based on the context that proceeded it?

In addition to the SOS-token, or Start of Sentence, there’s also the buffer token which Is just used to fill out the input and output if its under the context parameter.

~~Seq-Seq also uses a 3~~~~rd~~ ~~special token called EOS, or end of sentence, that went right before the SOS token; but I didn’t really see much use for it or difference then just an SOS token so I didn’t keep it.~~

15.)

Before even that I was trying to put together the best hyperparameters as I could for this decoder model, because one thing with models is that they get very big very fast. And it’s important to make them big enough to be powerful, but for my case where I have limited time, small enough to learn at a fast enough rate.

You’ll notice that my Decoder Transformer is actually smaller than the original example, and that’s because its vocabulary is 6 times smaller, but is bigger in different dimensions. In the By-Subword the vocabulary is the same, using tiktoken with 2 more special tokens and its 4.8x bigger.

I would consider adding more layers of the decoder and encoder when making the full transformer.

The Seq-Seq was very large mostly because a lot of its dimensions are based on the vocabulary size more so than even the Transformer. Still smaller then GPT-2.

16.)

Models are made of parameters, or weights and biases, and training a model is adjusting them to find the best values to get the best predictions based on giving it the correct input and output, through back propagation.

17.)

One of the special dimensions of the model or a Hyperparameters, being the “context” size of the model, was already decided to be 256 because that’s what I wanted the character limit for the Prompted question to be.

But determining something like how many sequences of that context should be processed at once, or batchsize, was something I had to determine. And the size under 32 that gave me the best training was 25 for the Decoder Model.

18.)

Some other things I tried then failed and learned from was how to best use RAM with large data, how Models are given Prompts and Answers to learn off of, trade offs between runtime and memory, and avoiding smaller context parameters.

19/20.)

Getting into end results:

One goal that spanned over the whole project was making my containers and classes like a product. And really expanding them out to be very versatile and Stable, because with all the different experimenting I was doing, being able to do things very quickly, easily, and modifiable made debugging and launching things very easy.

21.)

Also having a lot of helper functions and working on a scrum board, which I would have liked to show but they banned me for not paying.

22.)

All my training was done on the CRC, on a GPU node with 12 cores, A100 Partition. I just wouldn’t have been able to get this project half as far if it wasn’t for this resource.

I probably would’ve gotten better training by requesting more cores, and GPU cards, but I really wanted a dataset that was all the same because I was going to compare the total runtime of each model’s training but now, I realize it’s not that important. And I shouldn’t have held that back just because it’s what I started with.

23.)

With the CRC there was a lot of just daily upkeep, checking in with the CRC that my logs are still updating and didn’t run into errors and clearing out previous model saves to not take up too much space.

24.)

I had only finished the code for all 4 types of Decoder models two weeks ago, and the Seq-Seq one week ago, so I tried to train them as much as I could. Obviously, they’d be better with another two weeks, months, years. But let’s get into the end of this presentation.

25.)

So this is what my total pretraining of the Decoder Transformer using By-Character looked like, and I obviously made a mistake, one of the features of my container was skipping training on a piece of data once a target loss was hit a number of times, except I ran it with that goal at 0.7. So you can see it on the left over all of the tokens, or on the right and middle with each data piece overlayed that it just completely stops at 0.7 loss.

The issue is 0.7 loss is a decent amount of loss. There is a lot to still be learned there. It should’ve been set at 0.1, expecting to almost never hit that, or even 0 to remove this feature entirely. It was definitely a mistake, and was left in the By-Subword pretrain as well. But caught before SFT.

And this is what it generates given some context to run off of: calling me an idiot for good reason, but for by-character you can see it making complete words and sentences although spoken like they’re from 500 years ago.

This was my first Model that I ran for a very long time, I was really only training in small batches before, just making sure things were working. Where the skipping seemed like an improvement.

I think also a more curated dataset would’ve been better as the Gutenburg text files have some XML which I tried to cut out as best I could, also because its public domain the books are from very long ago, and some books are just kind of odd like one I saw was the human genome that I caught and cut out. But there could be more like it that I missed.

I was actually pleasantly surprised it didn’t generate a bunch of reference brackets which was an issue I saw a lot in smaller training practices.

I think with my mistake considering, this went pretty well.

26.)

So the training after pretraining, where instead of just raw text you train it off prompts and responses is called SFT or Supervised Finetuning.

For full Transformer models, this is just input the Question into the Encoder, and the Answer into the Decoder. But with working on the Decoder only, I just used the same approach as Seq-Seq which was to feed the Question Concatenated with the Answer between a special token, with any empty space filled by a buffer token.

27.)

I tried two different kinds of this SFT training,

type1 being: where both the Question and Answer are under 256 characters.

and type 2: where it was trained off of any size of question and answer.

The reason being that if it can only read between 256 characters as its context, and it needs the SOS token for their separation, will the fact that its always included be an issue or not when training or generating?

28.)

And looking at the training: For each new piece of data, it was having a lot of loss, then having almost no loss at all immediately after. So it didn’t seem to be retaining any information across each piece of data which an extremely big issue and proves why you need an encoder.

29.)

And looking at the Generation: You can see that it’s just all gibberish, this experiment failed. Even if the entire Question, SOS token, and Answer are within view or not; A decoder alone cannot pick up the nuance to produce an answer and is why the encoder module is needed.

30.)

And here’s the same thing but By-Subword instead of By-Character. Training Loss Graph there, same issue as before. It didn’t get as much training done because its vocabulary size was 6 times bigger than by-character.

It seems even more important that this pretraining data is curated for By-Subword because it gives a lot of brackets with some action happening like a movie script. It’s really cool it was able to pick this up, but not so useful for my purposes. Could also be a lack of more training.

I think this was mostly because in By-Character it’s able to train off the sentence structure between weird variations, and with more training these variations and bad pieces of data have less weight over the whole training process. But with By-Subword, the data is split up too widely and it can’t ignore something like that.

31.)

This SFT training loss: It looks almost identical to the By-Character. One thing to mention is that By-Subword was so much slower then By-Character because of its larger token dictionary that Type 1 didn’t even finish its 3rd dataset with the time I gave it.

32.)

And its final generation: Also gibberish. Type1 had some more frequent vocabulary akin to answering, likely due to missing that finally dataset which was conversation based. I thought it would be a good addition, but for future purposes I would leave it out.

Type 1 had some funny responses that seem proper, but are most definitely a coincidence.

Type 2 was more gargled and actually swore a lot.

33.)

And finally Seq-Seq, which didn’t need any pretraining. It can go right to Prompted Training.

This training loss had a strange uphill curve, but its training loss was extremely low the entire time.

Under 0.5 94% of every step.

34.)

Its generation: I honestly didn’t see much improvement from it in e ither training types from the original example I worked off of. Theres possibly something wrong with my container, I was in too much of a crunch to test it out as much as I did the other models.

35.)

To recap, over the summer I learned all about Large Language Models, the different Machine Learning Architectures involved first hand, had a lot of my original questions on the topic answered as well as all the questions I had come up throughout the project, and tried and failed quite a lot.

I wasn’t able to make a ‘Nice Looking Model’ with good enough output to call my original goal of making a working chatbot like Chat-GPT, which is kind of sad. Although maybe too ambitious.

Possibly I could have reached that goal if I cut out everything and only went for that but I don’t think I would have learned as much as I did if so.

Live demo if can fit it in

35.)

And that is my SURG 2024 Project. Thank you so much for watching, thank you to all the SURG members and mentors.

Thank you, James, for coordinating, and letting me back in to this program.

Thank you, Dr. George, for SHREC and giving me this opportunity.

Thank you, Eileen and Steven, for hosting the ML team.

And a thank you to Andrej Karpathy, one of the founders of OpenAI, who really was the foundation for everything I learned this summer. He has so many tutorials and resources for LLMs that I would’ve been completely lost in this project without.