Auto-Suggest Next Word

1. Summary

Predicting what word comes next is one of the fundamental tasks of NLP and has several applications. It is used by people every day, for example you while writing texts or emails. It helps in minimizing keystrokes which in turn saves time while typing, checks spelling errors and helps in word recall. Students who are still working on their language skills can take its help. Also, people with dyslexia can be benefited by this. Also, it has become an integral part in search algorithms like google search. According to a report by Markets and Markets [1], "The global Natural Language Processing (NLP) market size to grow from USD 11.6 billion in 2020 to USD 35.1 billion by 2026, at a Compound Annual Growth Rate (CAGR) of 20.3% during the forecast period." The report also states, "The rise in the adoption of NLP-based applications across verticals to enhance customer experience and increase in investments in the healthcare vertical is expected to offer opportunities for NLP vendors". This makes the problem of word prediction interesting.

Machines can generate new text using various statistical, probabilistic, and complex techniques called language models. These models are built to predict any probability of a pattern or sequence of words. For this project we explored how different language model architectures perform the job of text prediction the focus is on the n-gram language model, Long Short-Term Memory (LSTM) neural networks, and (Generative Pre-trained Transformer) GPT [2]. Language model. We used two different text corpus and have tried to understand how these other models perform on them and identify if there are some tweaks that can be made in the data or the model, that could make the performance better in the task of word prediction. Also, the same text data was used to evaluate the models which gave insights into how the models differ.

2. Data

The dataset that will be used is obtained from Project Gutenberg. It is a library of over 60000 free eBooks, from here we can get many stories, documentations, and text data which are necessary for our problem statement. The dataset has text files encoded in UTF-8 format. The data would be needed to be cleaned before use so that it is uniform. Also, they need to be split into sentences with start and end tokens, and new token for unknown words. For this project the following books are used,

- The Republic of Plato
- The complete works of William Shakespeare

The Republic of Plato contains 1427843 characters, and the complete works of William Shakespeare contains 5535477 characters. To prepare the data that will be suitable for working with the model we tokenized the data into sentences using NLTK tokenizer [6], resulting in total of 13558 sentences (255816 words) for Plato dataset and 101274 sentences

(961950 words) for Shakespeare dataset. Further the data was cleaned by removing 'String' punctuations and special tokens such as '\n', '\r', etc and the words were lower cased. Next the sentences with length more than 100 words or less than 5 words were dropped, bringing the final data to be of sentence size 9384 for Plato and 48000 for Shakespeare. Next each sentence was appended with start <s> and end </s> tokens. We split the data into 80% train set and 20% test set.

3. Modelling

In order to set up a baseline for the above defined problem we chose n-gram model with n equalling two, three and five. These are simple language models which calculate the probability of next word given a sequence of words and they just look at previous n-1 words to calculate the same. We have built the model such that in the train dataset if any word is of frequency one the same will be renamed to token <UNK>. Also, in the test dataset if the word is not in vocabulary or if the frequency was one in vocabulary it will also be tokenized as <UNK>. We also implemented smoothing as an optional parameter.

Now when coming to LSTM model, we used the TensorFlow package [5]. Here we are using a Stacked LSTM Model. This is developed by stacking a Bidirectional LSTM layer over a LSTM layer. The architecture is as follows:

Input Data -> Embedding Layer -> Bidirectional LSTM -> LSTM -> Dense Layer -> Output

We have coded three such models with sequence lengths 2, 3, and 5. The data is tokenized using the text processing library Tokenizer () from Keras and converted into the required sequences. The vocabulary considering the first 2000 lines of the Shakespeare data is about 4208. The model has a total of 353,166 trainable parameters. The inputs are converted into sequences and are padded to match the sequence length. To predict the probability of the output layer we have used the SoftMax Function. The number of units of Bi-LSTM and LSTM is 100, 50, respectively. The key concept behind this implementation is to retain the sequence information of the words in x to predict the next word class in y. We have divided the text data into train and test splits and evaluated its perplexity for all the sequence lengths. The perplexity reduced as the sequence length increased for both train and test splits.

The final model that we developed was a transformer-based model called GPT-2 from Open AI using Hugging Face library, it is made up of the decoder part of the Transformer architecture. It is a pre-trained model it was trained on a massive 40GB dataset called WebText that the OpenAI researchers crawled from the internet as part of the research effort. When fine-tuning GPT-2, we simply over-emphasize certain things that GPT-2 has already learned, making some word sequences more probable than others, also pushing GPT-2 to "forget" previously known connections, that are not important to the fine-tuning task. It works using a Byte-pair encoding technique.

To set up the architecture of fine tuning we first load the dataset using dataset package from transformers, by default the max length of sequence is 1024 for GPT-2. As our sentence length is small, we have used the average length of sentence in the corpus as sequence length. Also, to tokenize the data here we are using GPT2TokenizerFast, imported from transformer. It has a vocabulary of 50257 words. End of sentence token is used to pad the sequence. For both Plato and Shakespeare, the batch size was set to 8 and the epochs were set to 10. As the data will be character encoded, there won't be an issue of unknown words, in fact the model can generate new unseen words.

After the final model to predict the next word, the following approaches are used, Greedy Search - chooses the best possible next word based on highest probability from one hypothesis, Beam Search - chooses the high probability next word from n hypothesis, Random Sampling - chooses random next word from possible hypothesis, however as the temperature is set high, it will ignore low probability words.

4. Results

We used perplexity as evaluation metric for all our models. Given a model and an input text sequence, perplexity measures how likely the model is to generate the input text sequence. As a metric, it can be used to evaluate how well the model has learned the distribution of the text it was trained on. Perplexity is defined as the exponentiated average negative log-likelihood of a sequence. If we have a tokenized sequence, $X = (x_0, x_1, \dots , x\_t)$ then the perplexity of X is:

$$PPL(x) = \exp\left(\frac{-1}{t}\sum_{i1}^{t}(log p\theta(\, xi \mid x < i\,))\right)$$

For the n-gram language model, we wrote the code from scratch to calculate perplexity using the above equation. For the LSTM model, the exponent of the categorical cross-entropy loss gives the perplexity. We must use e instead of 2 as a base because TensorFlow measures the cross-entropy loss with the natural logarithm.

Model	Perplexity				Sequence Length
	Plato		Shakespeare		
	Train	Test	Train	Test	
gpt2 - pretrained	1241	692	3008	15042	25- Plato 35 - Shakespeare
gpt2 – Fine Tuned	7	67	860	1100	25, 35
LSTM - Stacked	194,144, 136	239, 197, 164	206,196, 174	248, 207, 182	2, 3, 5
N-gram	NA	348, 481, 295	NA	1200, 2113, 1316	2, 3, 5

Table 1: Shows how the perplexity of different modes compare among Plato and Shakespeare Dataset, along with the different input sequences of data the mode was trained and tested upon

The GPT model was evaluated using perplexity metric from evaluating a package of Hugging Face transformer [4]. This metric outputs a dictionary with the perplexity scores for the text input in the list, and the average perplexity. We are considering the average score here.

Table1 shows the Perplexity calculated for the two datasets. From the results it can be observed that the fine-tuned gpt2 model performs better on the Plato dataset, in fact when the same train data was used to calculate perplexity it gave an output of 6, which is impressive. But in contrast we can see that the model did not perform well on the Shakespeare data. This can be because the language and words that are used in the literature may not be known to the model previously. But with fine-tuning we see that the score has drastically improved. The same can be further improved by reducing the loss with more epochs and a decaying learning rate. But overall, the results are consistent and as expected higher the n-grams or the sequence length, the less perplexity value is. Also, we can note that Plato's data scores are better than Shakespeare, this could be due to the word complexity and restricted vocabulary.

Finally, as expected gpt2 performs better than the LSTM model, which in turn performs better than the n-gram language model. One notable thing is that for n-gram models the score remains same with or without smoothing. The LSTM model performance is significantly better considering that it only ran for 10 epochs and has only 353,166 trainable parameters. It performed better with significantly fast train times than the n-gram language model and close to the optimized GPT.

5. Future Works

The Scope of the project was limited to finding the next word given a sequence of words. But the sequence was limited to two, three and 5 words while training the models. We can use the models like LSTM and GPT to be trained on data of variable length. For example, GPT has a maximum sequence length of 1024. The Plato and Shakespeare data have maximum length of each sentence around 200 words and the average sentence length is around 20 words, this means we can use the entire sequence to train the model with padding, this may give better results as the transformer model works using attention mechanism and longer sentence means more context the model captures, and this can lead to better test results.

Another major improvement that can be done is to train the neural network models on a larger corpus and reduce the loss significantly to predict words with better and confident probabilities. This can be done using multi-core GPUs for training. Also, additionally the models can be extended to more than one word prediction that is, sentence generation or auto completion of sentences. This proves to be much more practical use case of natural language models. For example, we can fine tune the model with the data from specific user, where the data reflects the style of writing such as type of sentences and words frequently used. This may in turn provide us with a model that can mimic the sentences that will be written by the user and can used for autocompletion in email, essays and so on.

6. Works Cited

[1] https://www.marketsandmarkets.com/Market-Reports/natural-language-processing-nlp-825.html

[2] https://medium.com/geekculture/next-word-prediction-using-lstms-98a1edec8594

[3] https://medium.com/analytics-vidhya/next-word-prediction-using-swiftkey-data-f121f59bc7d

[4] https://huggingface.co/spaces/evaluate-metric/perplexity

[5] https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[6] https://www.nltk.org/api/nltk.tokenize.html