**Next Word Prediction**

**Introduction:**

**Next word prediction** involves predicting the next word. So given a sequence of words generated from the corpus, to predict the next word which has highest probability of occurrence. Thus, it is a predictive modeling problem for languages **also known as Language Modeling.** **This is the Probabilistic model.**

We can also approach this problem in another way. We can consider each of the next word to be predicted as a class. So, it **can be treated as Multi-Class Classification problem.**

Language modeling tasks predicts words in a sentence, making these types of models great at generating text. There are two types of language modeling, causal and masked.

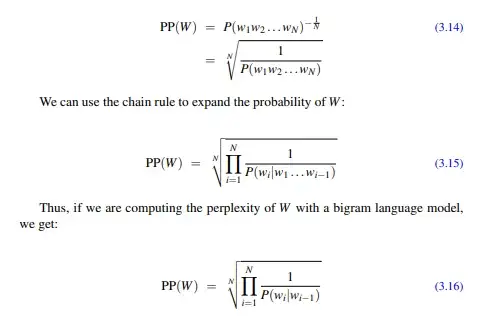
* **Causal language modeling** predicts the next token in a sequence of tokens, and the model **can only attend to tokens on the left**. This means the model cannot see future tokens.
* **Masked language modeling** predicts a masked token in a sequence, and the model **can attend to tokens bidirectionally**. This means the model has full access to the tokens on the left and right.

**Objectives**

* Cleaning text data from the corpus
* Creating Sequences from the cleaned data
* Building Neural Language Models to predict next word

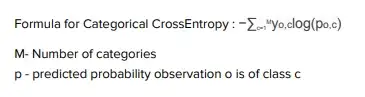
## **Performance Metric: For Probabilistic Model:**

## Perplexity: It is defined as the inverse probability of the test set normalized by the number of words.



**For Classification Model**

Categorical Cross Entropy: Each next word to be predicted is considered a category so will be using Categorical Cross entropy for this.



**Other metrics:**

Language model performance is measured by perplexity, cross entropy, and bits-per-character (BPC). The GLUE benchmark score is one example of broader, multi-task evaluation for language models [[1]](https://thegradient.pub/understanding-evaluation-metrics-for-language-models/#fn1). For more details refer the below link.

<https://thegradient.pub/understanding-evaluation-metrics-for-language-models/>

**Work-Flow:**

1. Data Collection
2. Sequence formation
3. Preprocessing: Removing white spaces, symbols, converting numbers to words, elaborating abbrevatives,etc
4. Tokenization
5. Word embedding
6. Model building/Importing
7. Training/Finetuning
8. Testing
9. Inference

**Deep Learning Methods:**

* **N-grams approach**

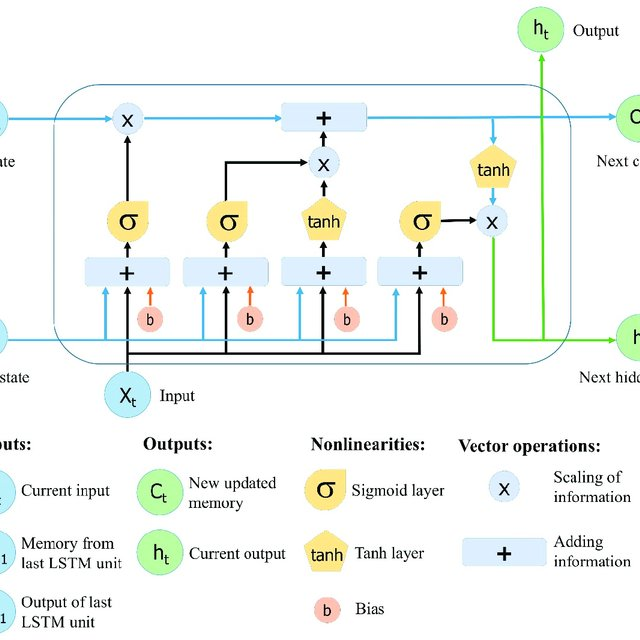
Focus on the ‘Markov Chains’ to predict the likelihood of each following word or character based on the training corpus. That is, they predict the next word given the previous work from the corpus.

**Limitations**: Markov chains do not have memory. So, it's hard to be able to remember the context of the information.

* **LSTM(Uni/Bi):**

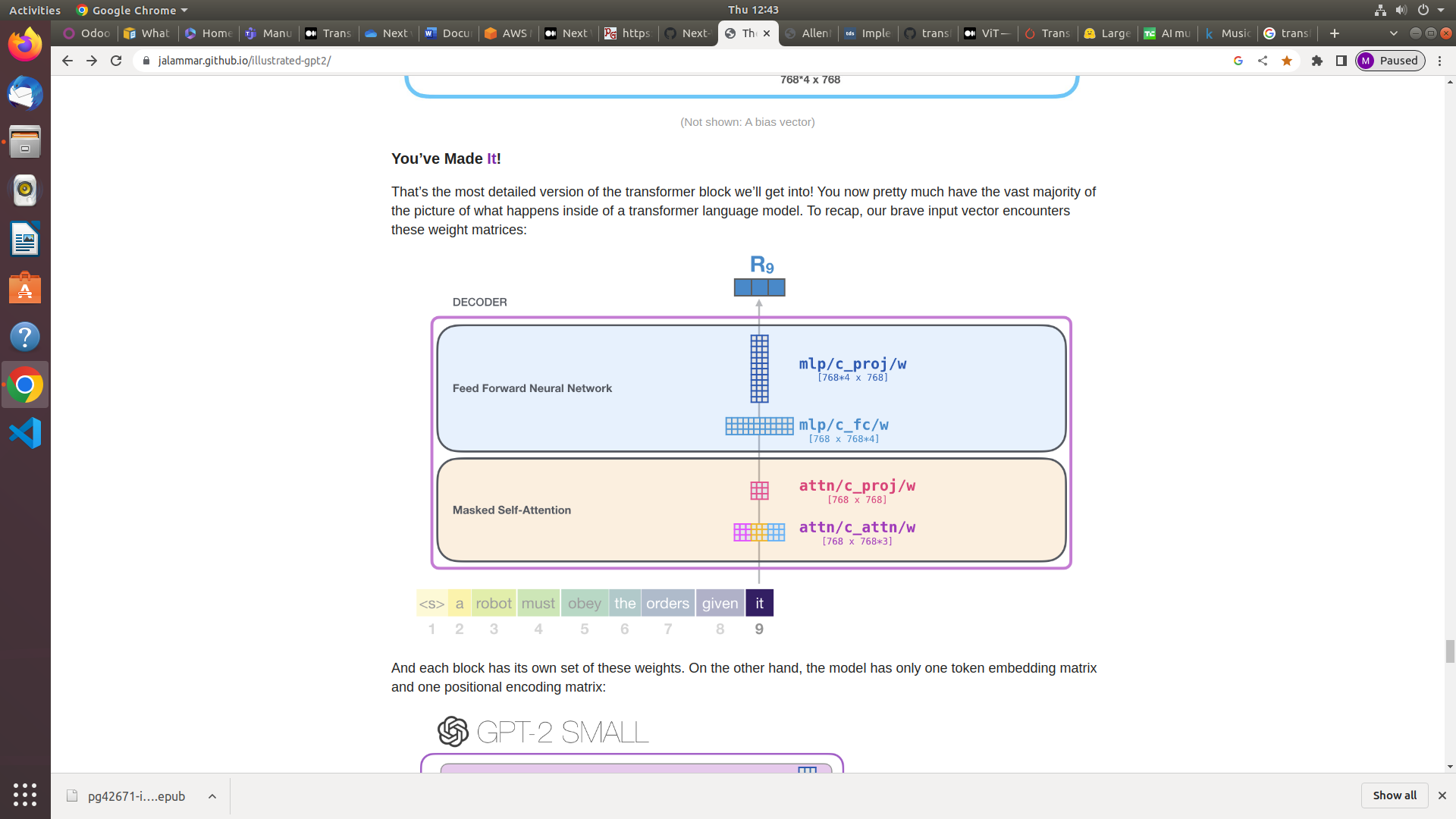
Standard RNNs and other language models become less accurate when the gap between the context and the word to be predicted increases. Here’s when LSTM comes in use to tackle the long-term dependency problem because it has memory cells to remember the previous context.

**Limitations**: Though it remembers the context is only efficient with small texts and also fails to understand references in a statement. That’s where the idea of attention comes in.



* **Transformer-Decoder:**

Has only decoder blocks of a transformer, with access to only the tokens on the left and the present word itself. GPT-2 uses this architecture to efficiently predict the next word. Currently the state-of-the-art models use this architecture.



**Preview:**

|  |  |  |
| --- | --- | --- |
| Key-points | Building the Model | Opting for pre-trained |
| Memory: Training | 64GB GPU (and above) | 16-32GB GPU (fine tuning) |
| Memory: Storage+Inf | 3GB + 16GB (approx.) | 3GB + 16GB(RAM) |
| Dataset | Wikipedia and books | Domain specific |
| Cost | Expensive | Low/Nill |
| Result quality | Requires lot of corrections and trials to get the desired output | End result can be finetuned easily |
| Working knowledge | High | Avg |

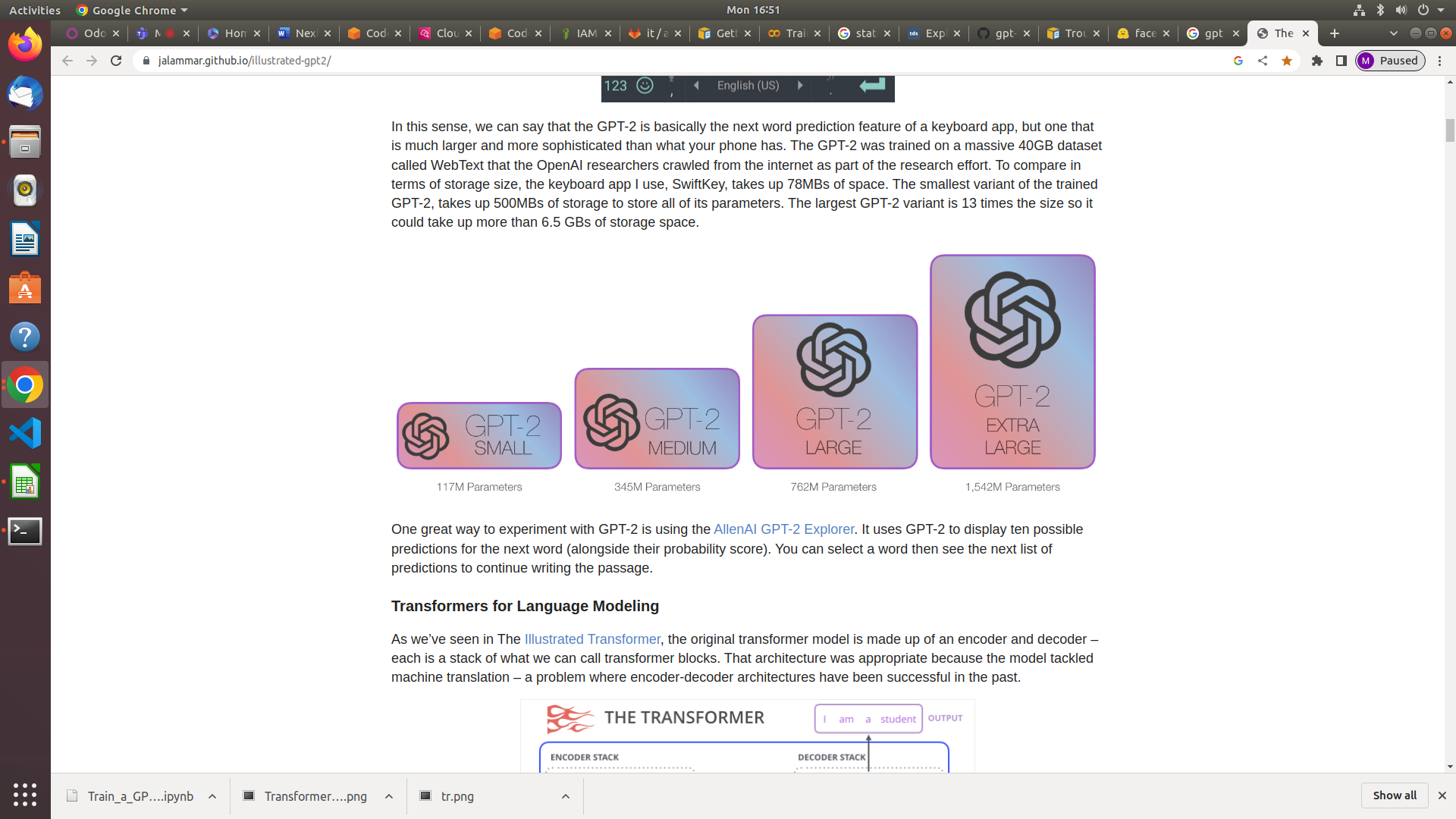
**Time Required:**

|  |  |  |
| --- | --- | --- |
| Jobs | Newly built | Pre trained |
| Data Collection | 10 days | 10 days |
| Text Preprocessing | 3 days | 5 days |
| Model building/importing | 5/20 days (varies on the architecture chosen) | 3 days |
| Training/Finetuning | 10 days | 10 days |
| Testing | 3 days | 3 days |
| Inference | 3 days | 3 days |
| Total days: | 34/49 days | 34 days |

**Note:** The “Chinchilla Point” is the level of data, as measured in tokens, that is required to train a model effectively and that converges to the right answer.

**Pre-Trained Models:**

* GPT-2: Generative Pre-Trained Transformer with the below variants. Refer the link for model card. <https://huggingface.co/gpt2>



* Facebook/opt: Open Pretrained Transformers (OPT), a suite of decoder-only pre-trained transformers ranging from 125M to 175B parameters. Refer below link for model card. <https://huggingface.co/facebook/opt-2.7b>
* BERT: Bidirectional Encoder Representations from Transformers is by researchers at Google AI Language. Trained on parameters ranging from 110-340M. Refer the link for model card. <https://huggingface.co/bert-base-uncased>

**References:**

<https://towardsdatascience.com/exploring-the-next-word-predictor-5e22aeb85d8f>

<https://wingedsheep.com/building-a-language-model/>

<https://www.nextplatform.com/2022/12/01/counting-the-cost-of-training-large-language-models/>

<https://pub.towardsai.net/run-very-large-language-models-on-your-computer-390dd33838bb>