Modeling of complex phenomena and networks by machine learning

Nurzhan Sarzhan

December 8, 2020

Abstract

Language modelling is nowadays one the most highly developing fields in Deep Learning. A new neural network architecture called The Transformer made a revolution in NLP and broadened horizons for future discoveries. Researchers around the World are working with this architecture in order to understand it better and improve it's performance achieving state-of-the-art results. So I wanted to be a part of this community and decided to train my own Language Model based on the existing one. I have fine-tuned a GPT-2 small on "A Song of Ice and Fire" novel books. It is able to generate new texts about this epic fantasy. As a result I've got a working script, which can produce sequences with grammatical and lexical structure and of course, on the topics about Game of Thrones. However, model often make mistakes in punctuation and misses context of the sentence. Anyway, it can be a good baseline for work with my Thesis in future.

1 Previous Work

On my previous modelling project work, I was working with text classification problem. My goal was to create a model, which is able to detect tweets written about disasters. The idead was to create a program, which is able to detect disasters in the city by scanning tweeter in real-time. The base architectures I used, was BERT (Bidirectional Encoder Representations from Transformers), proposed by Google team, which was a SOTA algorithm in 2019 and achieved top scores on various text tasks. Moreover, it was really easy to use it for own projects. On my model (text classifier with a BERT layer), it also achieved better results than previous known algorithms, such as classifiers with BOW, TF-IDF, or even recurrent neural networks (RNNs, LSTM, GRU). This project helped me to dive in the sphere of Natural Language Processing with Deep Learning and open possibilities for working with more complex and interesting tasks such as language modelling.

2 Introduction

On this modelling project work I've decided to continue work with Transformers, but on different NLP task - text generation. I've stopped on it because of few reasons:

- Transformer is still the best architecture for NLP problems and it's still not fully studied. Therefore, there are dozens of possibilities for discovering new features of it.
- Text generation is highly demanded nowadays. We have various virtual assistants (Siri, Alexa, Alisa, etc) and chat-bots, where this technology is used. They are being developed with incredible speed and still haven't reached their limits.
- Text generation model is a crucial part of the projects, which I would like to develop in future such as Q&A bots for banking apps and own assistant, which can do things limited by current existing assistants.

After choosing problem I'm going to work with, it was necessary to decide, which architecture will be appropriate. Currently, there are many Transformer architectures, which can be used for text generation. GPT-2, OpenAi-GPT, CTRL, XLNet, Transfo-XL and Reformer are mostly known examples. My decision was GPT-2, a large-scale unsupervised language model developed by OpenAi team. This model achieved state-of-the-art performance on various text modelling benchmarks and text generation as well.
And finally, I came to dataset selection. Based on my own interests and GPT-2 requirements, I've chosen books of a Song of Ice and Fire novel by George Martin. I was really interested in how could computer write a book based on it's history and is it possible to finish the series.

3 Literature Review

Literature played an important role in my research, I've read dozens of articles from Medium, Hugginface documentation and github pages of various researchers. However the most influent ones are:

- A paper "Attention is all you need" [7] written by Ashish Vaswani et al where authors introduced Transformer's architecture was the most important source in my project work and probably not only in my work, but also in papers of other deep learning researchers of last years. This architecture, based on alread know mechanism called Attention, made a huge revolution in Natural Language Processing, because it doesn't suffer from problems which recurrent neural networks (RNNs) had such as gradient vanishing and exploding problems, and complexity of parallel computing.
- The second source I used is paper about GPT and GPT-2 [5]. These architectures don't have big differences except of sizes and datasets used for training. GPT-2 is much bigger than GPT and it was trained on bigger dataset (WebText against Book Corpus). Anyway, these papers helped me to understand architectures in details and distinguish from base transformer architecture. Moreover, GPT makes it clear that "in Transformers, bigger is better" and outstanding results can't be achieved without huge datasets and unlimited computational resources (people currently are trying to prove that this statement is wrong. I hope they will do it).
- Book "Deep Learning" written by Ian Goodfellow, is truly one of the best books about neural networks. Thanks to it, I understood mathematics, which are working under the hood, and it helped me to improve my overall skills in deep learning.
- Article "The Illustrated GPT-2 (Visualizing Transformer Language Models)" by Jay Alammar [1] revealed to me a secret of Attention mechanism. He succinctly described architecture of GPT and mechanisms such as self-attention, masked self-attention, multihead attention of so on.
- Philipp Schmid's tutorial about fine-tuning GPT-2 on custom datasets [6] helped me in building my own text generator with using Huggingface's Transformers library.

4 Historical Background

4.1 Recurrent neural networks

Historically, recurrent neural networks (RNN) (Figure 1), long short-term memory (LSTM) and gated recurrent units (GRU) were state of the art models and achieved incredible results in sequence modelling problems like language modelling [2]. Since then, lots of attempts were made to increase their efficiency by applying various techniques. All of these architectures had common mechanisms of the work under the hood:

- **sequential processing**. Sentences were processed tokens by tokens
- Information of the past was gained through **hidden states.**

First property was the reason, that RNNs and LSTMs were unable to be trained in parallel. Because in order to calculate a vector space for n-th token, we need to know vector space of the $(n-1)$-th token, which we also need to compute. Thus, parallel computing doesn't seem possible in this case. Following from this, training of the model, which can give more or less appropriate accuracy, could take a lot of time. The reduction of this fundamental constraint of the sequential computation has been the main goal for NLP researchers.
The second biggest disadvantage is related with vanishing/exploding gradients. This means that as length of sentence growth the cost function of the model can immediately explode to huge values (explode) or drop to zero (vanish) only because of inappropriate initialization the model's weights. Therefore, further training of the model will be impossible. In other words, the model will be biased by most recent inputs in the sequence and older inputs will have practically no effect in the output at the step.

4.2 Transformers

4.2.1 Attention mechanism

In 2015 Bahdanau et al proposed a revolutionary mechanism [3], which became a one of the most valuable breakthroughs in Deep Learning research in the last decade. It is called Attention. His neural machine translation was

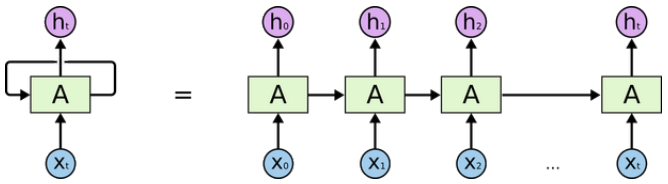

Figure 1: Basic RNN architecture

based on encoder-decoder RNNs/LSTMs in pair with attention. Here is diagram of the model from Bahdanau's paper: However, this model did not achieve impressive results, because recurrent neural networks were still be using

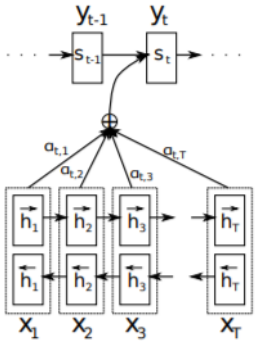

Figure 2: Attention mechanism connected to LSTM units

together with their drawbacks, however next researchers tried to eliminate it.

4.2.2 Attention Is All You Need

Hopefully, 2 years later, in 2017, Vaswani et al in the famous paper "Attention is all you need" stated that attention is self-sufficient mechanism for working with text data and models with it does not require any RNN in it's architecture. Additionally Attention was redefined by providing a very generic and broad definition of Attention based on key, query, and value vectors. And also Vaswani et al proposed another concept called multi-headed Attention. And these proposals became a biggest contribution in language modelling.
New architecture proposed by Vashani called Transformer (Figure 3) was a first transduction model which relied entirely on self-attention layers. It is the encoder-decoder neural network, where encoder part maps an input sequence of symbor representations $(x_1, x_2, ..., x_n)$ to a vector $(z_1, z_2, ..., z_n)$, while decoder auto-regressively generates vectors $(y_1, y_2, ..., y_m)$ at a time $i \in 1, 2, ..., m$, which then are decoded to corresponding tokens. Both encoder and decoder parts of the Transformer are stacks of N identical layers, which are compositions of multi-head self-attention mechanism and a simple position-wise fully-connected feed-forward neural network.
Then, after training on the WMT 2014 English-to-German dataset (roughly 4.5 million sentence pairs), Transformer reached 28.4 BLEU score, which was a new state-of-the-art score which was more than 2.0 BLEU ahead of the previous models.

4.2.3 GPTs

This event motivated other deep learning researchers to develop their own variations of Transformer and apply them for Natural Language Understanding (NLU) and Natural Language Generation (NLG). In this way, dozens of Transformer architectures pretrained on more than 100 languages were developed only in 2 years. The most essetial of them are BERT (by Google Research team), GPT, GPT-2 and GPT-3 (from OpenAI), CTRL (from Salesforce), T5 (from Google AI), XLM (by Facebook) and so on. Each of these architectures has it's own history and can solve different NLP, however they will not be disclosed in this paper except of one. A family of GPT models.
OpenAI's GPT model was proposed by Alec Radford et al in the paper "Improving Language Understanding by Generative Pre-Training", which is a unidirectional transformer pre-trained on a large text corpus (the Toronto Book Corpus) with long range dependencies. GPT belong to auto-regressive models, i.e. it generates one token at a time. GPT's architecture consists only from decoder part of the initial Transformer architecture. On the

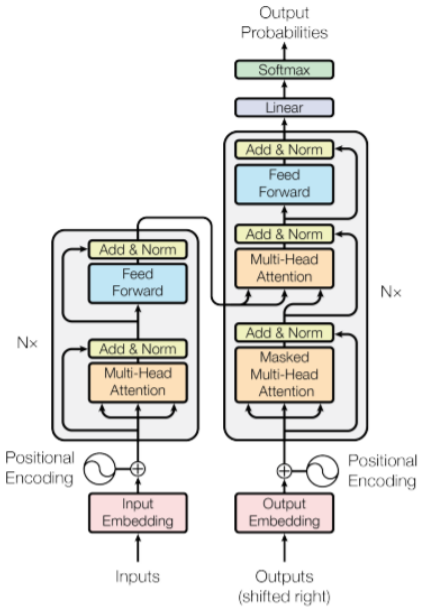

Figure 3: The Transformer - model architecture

Figure 4 you can find an architecture (**left**) and training objectives used in the work (**right**). After a while OpenAI

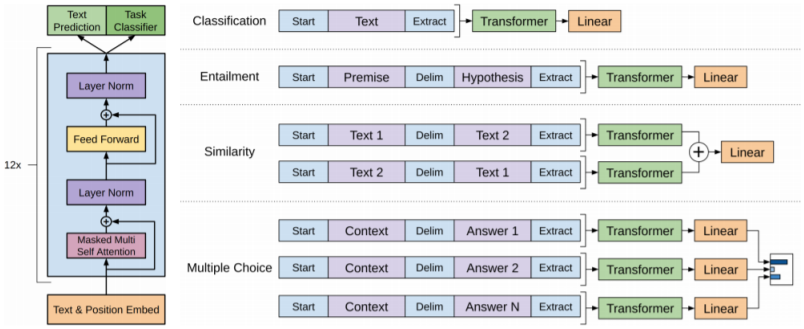

Figure 4: GPT architecture

team introduced new version of this model - GPT-2, which was trained on bigger dataset (more than 40GB of Internet text). Actually they released family of models, which differ only in sizes of networks, i.e. number of decoder stacks in it and the dimensionality of input vectors. They are small, medium, large and extra large with 117M, 345M, 762M and 1.5B parameters respectively (Figure 5 for comparison). Enlargement of datasets and networks size played it's role and it's scores on various tasks (Winograd Schema, LAMBADA and others) broke the record and became state-of-the-art model at one time. Hopefully, artificial intelligence research team from San Francisco didn't stop on the model's enlargement. In May 2020, OpenAI introduced the 3rd generation of their autoregressive language model - GPT-3 with an outstanding capacity of **175 billion** machine learning parameters (Figure 6 for comparison with other models), **96** decoder layers and **12288** token input vector space, which made it the biggest pre-trained model of currently existing ones. GPT-3 was described as one of the most interesting and important AI systems ever produced by David Chalmers, an Australian philosopher. Also, text generated by GPT-3 is really hard to distinguish from human written text, researchers and engineers of OpenAI were worried that their model is potentially dangerous and can used for malicious text generations. However, for today usage of GPT-3 is limited and can be accessed only via its API. Moreover, running such a monster requires enormous computational capabilities. Therefore, I've decided to work with the second generation of GPT, especially the version with 117 million parameters.

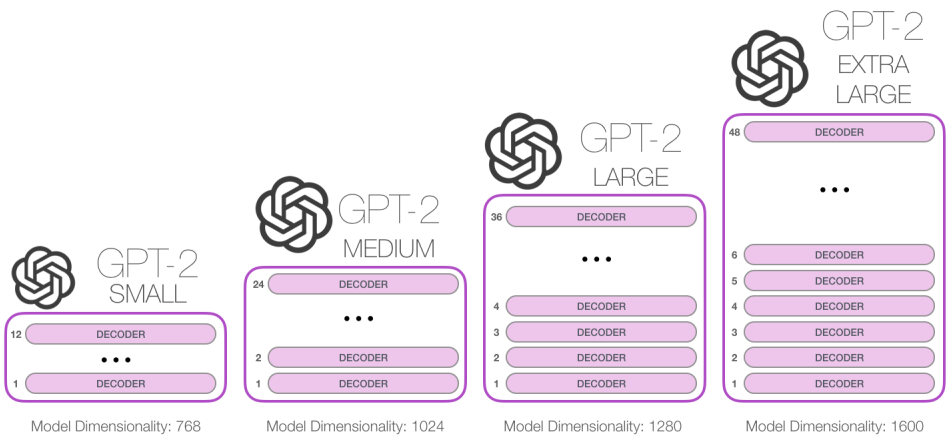

Figure 5: Types of GPT-2

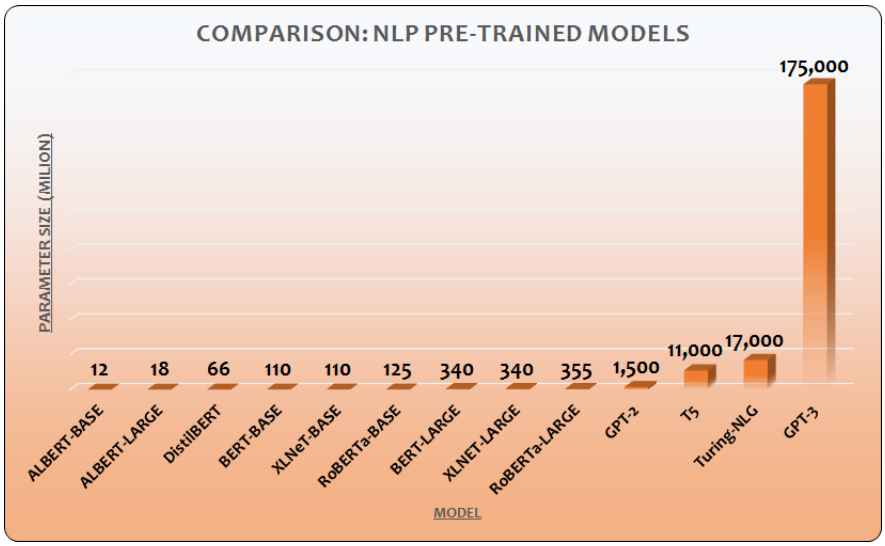

Figure 6: Sizes of current models

5 Theory

5.1 Language Modelling

According to Wikipedia definition, language model is a probability distribution of word over sequences of words $P(w_i|w_1, w_2, ..., w_{i-1})$, where i is a length of the sequence and w_i is a next word in the sequence. Generally speaking Language Model (LM) can be defined as a machine learning, which based on a given part of the sentence can predict next word in this sequence. Such models can be seen on modern smartphone keyboards, where most probable words are suggested next based on what user already typed. Though, language models are rarely used directly, they play a crucial role in other real-world applications. Text generation is one of them, which is basically auto-regressive launch of LM with some stopping criteria. So, mathematically speaking, language models are kind of Markov chains, where i-th word is dependent on the previous $i-1$ words. And our goal is optimize model's parameter in the way that most appropriate words would have a bigger probabilities.

5.2 Evaluation

Rises a question: how actually to measure performance of the model? Unfortunately, simple accuracy doesn't satisfy for this kind of NLP task, because two similar sentences in meaning, produced by a model, could be different by the construction. In reality, we have 2 approaches to evaluate and compare language models:

- **Extrinsic evaluation**. It involves evaluating the models by employing them in an actual task and looking at their final loss/accuracy. This is the best option as it's the only way to tangibly see how different models affect the task. However, it can be computationally expensive and slow as it requires training a full system.
- **Intrinsic evaluation**. This approach involves finding some metric to evaluate the language model itself, not taking into account the specific tasks it's going to be used for. While intrinsic evaluation is not as "good" as extrinsic evaluation as a final metric, it's a useful way of quickly comparing models.

The best language model is one that best predicts an unseen test set

Perplexity - inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1, w_2, ..., w_N)^{\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1, w_2, ..., w_N)}}$$

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, ..., w_{i-1})}}$$

Perplexity is an intrinsic evaluation method. Since we're taking the inverse probability, a *lower perplexity* indicates

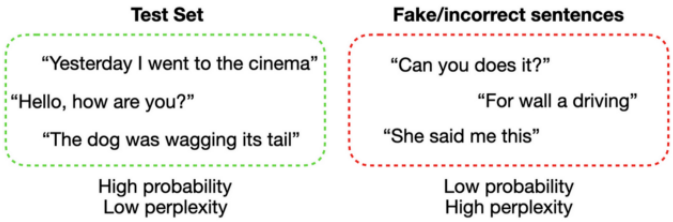

Figure 7: Examples of perplexity scoring

a *better model*, so it's necessary to have a baseline model, which gives a upper bound for perplexity on test dataset. Perplexity can also be defined as the exponential of the cross-entropy:

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N}\log_2 P(w_1, w_2, ..., w_N}$$

which is equivalent to the first formula.

5.3 Tokenization

We start any NLP modelling with a preprocessing of the raw corpus of text. Crusial part of it is tokenization: method of splitting sentences in into list of tokens (characters, words, word pieces, etc.). At first sight, it seems to split them by space character. For example, sentence "This is a cat" will be tokenized into [This, is, a, cat]. However, similar words can be splitted resulting different tokens. "Ilon Musk is a founder of Tesla company. Musk's company was founded in 2008" here, simple word tokenizer create 2 tokens for 'founder' and 'founded', however they have the same root.
Therefore, today in language modelling are commonly used 2 more complex tokenizers: **WordPiece** and **byte-pair encoding BPE**. The second tokenizer is used in GPT-2. These methods work similarly and they will split word "embedding" into tokens [em, ##bed, ##ding], where ## means that this token is a subword. The biggest advantage of these methods is that it can tokenize any word in the corpus, even if it was not represented in a tokenizer's dictionary.
Speaking about byte-pair encoding, it is a simple compression algorithm firstly introduced in 1994 by Philip Gage, which used in almost all modern NLP models today. The idea of compression is replacement of common pairs of consecutive bytes with that doesn't appear in the data (Figure 8). subword tokenization is performed slightly different: the frequently occurring subword pairs are merged together instead of being replaced by another byte to enable compression. So word *athazagoraphobia* will be tokenized into tokens [*'ath', 'az', 'agor', 'aphobia'*].

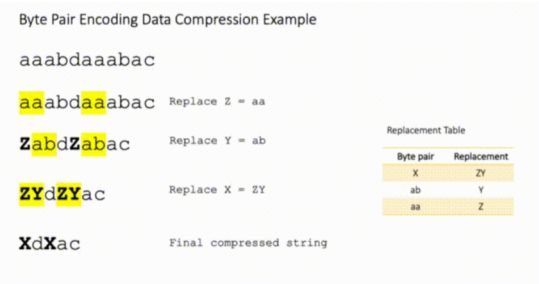

Figure 8: Byte-pair encoding example

5.4 Vectorization

After choosing evaluation metric and tokenizing input sentences, we can start modelling. However, at the beginning we have a list of tokens which is somehow needed to be represented as a number (or vector of numbers). We can simply assing each word in the text some unique id. However, the dictionary of this corpus can be enormously huge. As another option we can count how much each word occurs in the corpus and map to each word its frequency. This method is called Bag of Words (BoW). Or we all can apply more complex algorithm term frequency-inverse document frequency (tf-idf), which is originally a measure of originality of a word by comparing the number of tomes a word appears in a doc with the number of docs the word appears in.
Still, the mapping method have a hude disadvantage: they don't capture the meaning of words or in other words, tokes which similar to each other by meaning will be mapped to a vectors, which are completely not similar to each other. Thus, we come to word embedding, a neural network, which can vectorize tokens in the way, where similar words will be close to each other in a vector space as well (Figure 9). These networks are build on the idea, that "the word is defined by its neighbors" and similar words should have similar neighboring words. GPT-2 exactly

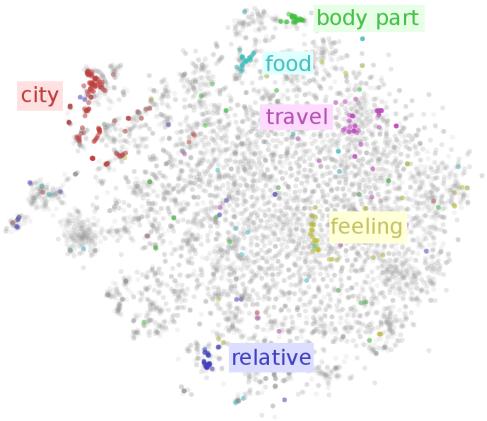

Figure 9: Example of how words can be embedded to 2D space

works with word embeddings, however it uses modern token embeddings algorithm, called ELMO [4].
Additionally with token embedding, NLP models use positional embeddings as well, in order to understand not just meaning of the word but also it position's influence. Positional Embeddings are introduced for recovering position information. There are two commonly used versions of postional embeddings: learned positional embeddings and sinusoidal positional embeddings. Both produce similar results.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

These 2 types of embedding are just summed to each other giving final embedding.

5.5 GPT-2 Architecture

As I mentioned before, in this project work I've used the smallest version of GPT-2, so the whole theoretical and practical will be to this architecture, though other versions differ from smallest one only by their sizes.

The GPT-2 is built with only Transformer's decoder blocks because of it's auto-regressive nature and each token is produced at a time, added to the sequence of inputs for further production of next tokens (Figure 10). Let's take the first law of robotics as an example for explaining:

A robot may not injure a human being or, through inaction, allow a human being to come to harm.

At first step, when part of the sentence is included, it predicts next word, while on the second step, it will add a predicted word to the input and then run step 1 again: Let's look deeper into GPT-2 (Figure 11). Each decoder

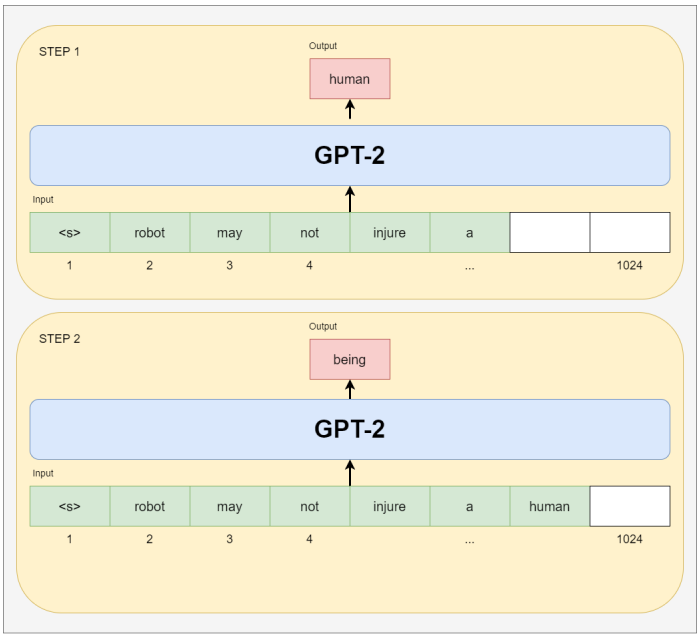

Figure 10: GPT-2 Auto-regression mechanism

block in the model consists of Masked Self-Attention Layer and Fee Forward Neural Network connected to it. Then all 12 blocks are stacked together and final decoder produced prediction for the next word in the sequence.

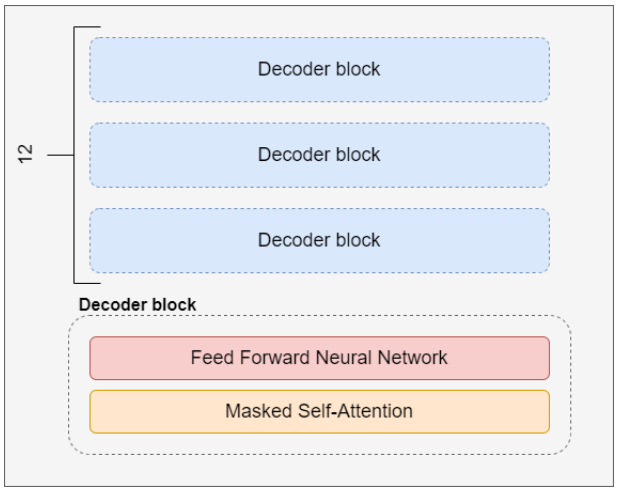

Figure 11: GPT-2 Arhitecture

Attention mechanism is simply next formula:

$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where W, K, V are Queries, Keys and Values matrices respectively.

6 Practice

6.1 Dataset

Hopefully, text generation task does not require specific datasets for feeding it to network. For example, text classification need a labeled texts (sentiment or some specific topic for each sequence), text summarizing needs pairs of sentences (first is a paragraph and the second is short sentence, which describes first one). Here, we need only a huge plain text. So we can use books, programming codes on github, songs, magazine articles, tweets and so on. GPT2 will be able to process every type of text corpus.
However, there are some recommendations on choosing datasets. As I didn't have a huge computational resources and training GPT-2 from zero requires enormous amount of time and computational power, I could only fine-tune an already pre-trained model. But which one to choose? Currently, on Hugginface.co there are dozens of already pre-trained GPT-2 models (on different languages, text corpuses and so on). Nonetheless, my decision was original pre-trained GPT-2 by OpenAI (on 40GB of English internet texts of general purpose).Thus, my dataset should be:

- Written in English
- Non-specific or General purpose. Programming codes are not appropriate in this case

Based on these requirements and also on the fact, that I am a huge fan of "A Song of Ice and Fire" series by George R. R. Martin, mostly known by it's TV adaptation "Game of Thrones", I decided to choose them as my dataset. I wanted to see how my fine-tuned GPT-2 could generate interesting lines, knowing a whole history of the Seven Kingdoms.
Technically speaking, dataset consists of about 83.000 sentences and it's total size is 4.5MB, which is sufficient for my small project. I've randomly splitted corpus into training and testing datasets, giving 20% for a test part.

6.2 Training

As I said before my pre-trained model is a GPT-2 small which has next parameters:

- It can process 1024 tokens, however it's dimensionality is 768
- Achitecture consists of 12 decoder blocks
- It's tokenizer is based on byte-level Byte-Pair-Encoding
- Pre-trained on 40GB of Internet text.

I've used free Google Colaboratory servers for fine-tuning, which offers Tesla T4 GPU and 13GB of RAM for training.
Next step is hyperparameters for training:

- Number of epochs = 5
- Training batch size = 32
- Learnining rate = 1e-5
- Optimizer = Adam with weight decay (AdamW)
- Sentence size in dataloader = 128

Training continued for 30 minutes loss on the training dataset was **3.2998** and on the validation set = **3.0993**

6.3 Results

Since perplexity is commonly used metric in language modelling, I've calculated it on the test set and it was equal to **21.2652**. To compare it's performance, I've calculated perplexity on non fine-tuned GPT-2 model and it gave **37.4395**. This means that our model gave an improvement of **76%** comparing to base model.
And finally, the most interesting part of the project - inference. For generating text from fine-tuned GPT-2 we just need to provide starting sequence of word to make generations more controllable or just generate random sequence. Let's look at most interesting results I've got:

> *Tyrion Lanister's daughter came crying and the king asked if she had been frightened, but the Lannisters let her out.*
>
> *Tyrion's face had turned pale at the cold touch of the blood inside.*
>
> *Arya Stark sent his brother to inspect the remains of her husband's body.*

At least we know now that Tyrion had a daughter who was kept locked up by his family and Arya had a husband, but he probably is not alive anymore.
Although it sometimes can give interesting results, more often fine-tuned model generates text with grammatical mistakes, produces some strange symbols and doesn't finish sequences (stops on the half of the sentence). Of course, it would be impossible to generate a text indistinguishable from a human speech with smallest GPT-2 and on dataset which weights only 5MB. So, there are plenty of plans for future work in improving this baseline

7 Conclusion

Finalizing my modelling project work, I would like to say that it was a long journey, because to understand how such complex algorithms work, I've learned in theory and in coding as well. Machine learning, neural networks (FFN, RNN, LSTM), natural language processing, language modelling, word embeddings, tokenization, attention mechanism is a list of topics which are themselves complex and require a deep knowledge. Moreover, coding is also important part in the project and knowledge in Python, Pytorch and Huggingface are vital in the work.
Through a thorough study of these aspects, I was able to build a language model, which can generate interesting lines in the context of Game of Thrones books.
All results of the project you can find in my GitHub page by this link

8 Future Work

My plans for future work are still related to Transformers. I'm going to explore another popular architectures

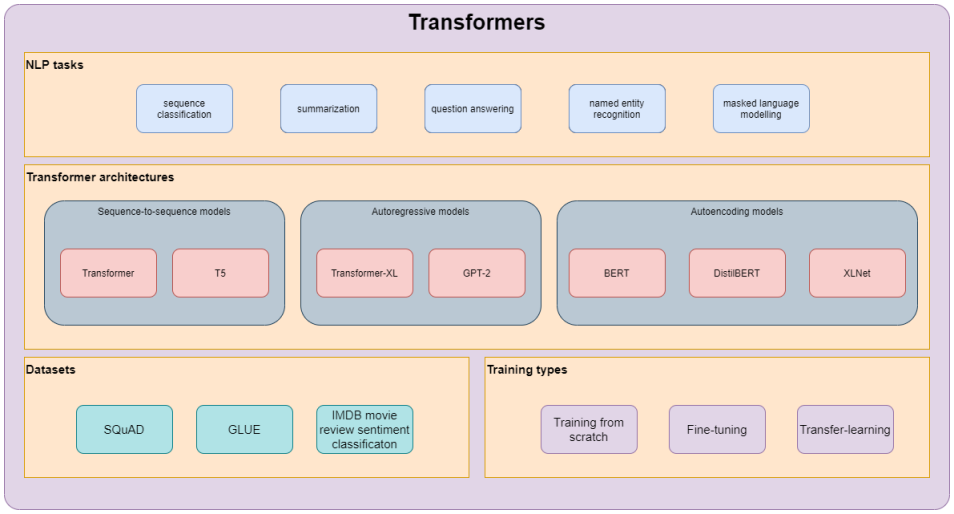

Figure 12: Plans for work with Transformers

(CTRL, XLNet, DistilBERT, BART and so on) and apply on different NLP tasks (text classification, summarization, question answering) with different training types and finally, compare with each other.

References

[1] Jay Alammar. "The Illustrated GPT-2 (Visualizing Transformer Language Models)". In: (Aug. 2019).

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *arXiv e-prints*, arXiv:1508.04025 (Aug. 2015), arXiv:1508.04025. arXiv: 1508.04025 [cs.CL].

[4] Matthew E. Peters et al. "Deep contextualized word representations". In: *arXiv e-prints*, arXiv:1802.05365 (Feb. 2018), arXiv:1802.05365. arXiv: 1802.05365 [cs.CL].

[5] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: ().

[6] Philipp Schmid. "Fine-tune a non-English GPT-2 Model with Huggingface". In: (Sept. 2020).

[7] Ashish Vaswani et al. "Attention Is All You Need". In: *arXiv e-prints*, arXiv:1706.03762 (June 2017), arXiv:1706.03762. arXiv: 1706.03762 [cs.CL].