GPT-2 Based Story Teller

CTEC3451 Development Project

De Montfort University

Artem Bobrov (P2547788)

May 5 2022

Contents

Abstract

The project described in this report is a story-telling application that uses the GPT-2 model to generate text. The model is trained on multiple datasets and is able to generate text from a variety of topics in the backend. The frontend is a web application that allows users to endlessly generate stories accompanied by thematic pictures. The point of the project is to prove the concept that neural networks, if trained properly, are able to generate data that is very similar to human language.

Introduction

Neural networks weren't so popular in the early days of computing, but they are now a common tool for machine learning. First neural network was invented by Frank Rosenblatt in the late 1950s (Rosenblatt 1958). Perceptron has only one layer of neurons, but more advance neural networks can have multiple layers. Before Transformer neural networks there were many different types of neural networks (e.g. Convolutional, Recurrent, LSTM, Feed Forward, etc).

Transformers were introduced in 2017 by Google (Vaswani et al. 2017) and are a new type of neural network. It uses self-attention mechanism to focus on the most relevant information in the input. It weights the significance of each part of the input differently.

This report focuses on the GPT-2 model, which is a transformer type neural network developed by OpenAI in 2019 (Radford et al. 2019). Despite the fact that GPT-3 is a newer model, it's predecessor is still a popular choice for text generation.

An application will be developed that utilises the features of GPT-2 and wraps it up in a user-friendly web interface that allows users to read freshly generated stories accompanied by thematic pictures. The development process will be examined and the results will be presented and discussed in depth. The usage of the key elements and techniques used both in backend and frontend will be explained. Moreover the project development and research cycle will be reflected on and other approaches that possibly could have been used to achieve different or similar results will be presented.

Objectives and Goals

Initial approach

It is important to state that the original plan for the project was to develop a similar application that generates pictures with attached text to them (known as 'memes') and allows users to watch them using a web interface. It has been planned to use different approach to build a repository of training data using 'Optical Character Recognition (OCR)' (Kay 2007) and 'Pillow' library to attach processed text to the pictures. It has been decided to adjust the original plan and change the topic from memes to stories.

New approach

The new approach doesn't require any massive changes to the original plan and the project will be developed using the same techniques - indetical frontend and almost the same backend. The only difference is that OCR will be removed from the backend because it's not needed for the project anymore. To minimize the deviation from the original plan, there will be an additional section in the web interface that creates 'memes' based on novels and other piece of art.

Reasons

Despite the fact that many of the features used in the initial approach have been already developed - Tesseract OCR, Pillow Image placing, 'Meme' scraper - one major issue made it extremely perplexing to continue the project without changing the approach.

The first puzzling part was hidden in reading the text (using Tesseract OCR) from the scraped images and translate them to text strings. Memes contain text captions that makes it easy for a human to read, but this same process is hard for machines to do so - especially via OCR. There were some images successfully processed by it but it didn't satisfy the requirement because the the success rate was too low. Many solutions were tried, but even pre-processing the images before feeding them to the OCR engine didn't seem to give much improvement. The second complexity was to build a meme repository itself. Using the meme scraper is not the best solution because the content obtained by it can be anything. Unfiltered, uncensored - it's a huge ethical problem.

General Description

Two general components of the development project are described in two sections. Such apparent separation between these categories is due to the fact that this approach allows a simpler, faster and more effiecient implementation with better user experience, it also improves code readability and maintainability.

The first one is the backend, which is responsible for generating text based on given datasets (stories, novels, other pieces of art, etc). It is implemented in Python and uses numerous libraries to achieve the goal. Multiple steps are involved in generating simple text samples including hours of training.

The second component is the frontend, which is a web application that creates a user-friendly, simple interface to scroll and read stories. It is implemented in JavaScript using many libraries, as well as the Python part. Such approach to frontend was chosen because web development is flexible and it's easy to adapt to different platforms. Also it's possible to extend it to a web service that doesn't require downloading the application to user's computer.

The goal is create a system with pre-trained model, which the user can use to scroll through the stories and react to each story leaving a positive or a negative feedback - using the 'Like' and 'Dislike' buttons. Afterwards, these reactions will be stored in a database to be used by the developer to understand which stories should be corrected and improved. The feedback system is designed to create a reflection on the pre-trained model as a whole and not on separate elements (stories). Given the feedback, the developer can decide to improve the model or not.

Major Components

Backend

The backend of the project is developed using Python programming language since it's a very popular language for machine learning and has a lot of libraries that can be used to achieve the goal. The backend is developed using PyCharm IDE, which is also a popular editor for Python. Calculations that are done in the backend can be executed on any machine with Python installed. The only issue is the optimisation - text generation requires a lot of computing power.

GPT-2 and the library

GPT-2 is the version of the GPT family used in the project. To operate with the network it was chosen to use a library called 'GPT-2-simple' (Woolf 2021) since it features an easy approach to control and train the model. The library is written in Python and offers a documentation which was used in order to work with the neural network. GPT-2-simple offers two types of models - one with 124 million parameters and one with 324 million parameters. Acknowledging the computational limits of the processing unit used - graphical (GPU), the first model with 124 million parameters was used. It is possible to use the second model but it requires a GPU with significantly more processing power and memory.

It is possible to use techniques that allow to save memory and train bigger models using memory saving gradients - the main memory loss in the computation is due to calculating the gradient of the loss by backpropagation. At the cost of one additional forward pass, the memory consumption can be reduced significantly (Chen et al. 2016). However Tensorflow framework (version 2), which is used by GPT-2-simple, is not optimized for such tasks.

Despite providing simpler control functions, this library also provides a lot of tweaking options. For example, some of the hyperparameters can be changed to improve the model performance during testing. Learning rate, batch size, number of training steps, number of accumulated gradients and optimizer can be changed. Number of layers and other more complex parameters can also be changed but in order to do that, the parameters must be examined and adjusted in Tensorflow code files.

The library also provides a way to load the model from a checkpoint file. This is useful when the model is trained in multiple steps and it's needed to continue training from the last checkpoint without losing progress.

Another important feature for testing and optimising the output is the possibility to use different optimisers. 'Adam' and 'SGD' are available optimisers. The first one is the most popular and is used in the project because the learning time is relatively short and the memory requirements are little (Musstafa 2022). These two aspects are crucial and thereforce the choice fell on 'Adam'.

Datasets

Several datasets are used in the project. It has been chosen to use three datasets to work with. The first one is the novel ‘Alice in Wonderland’ by Lewis Carroll (Johnsen 2013). This version was taken from GitHub Gist. It’s more adapted for machine learning than the original version since it’s shorter and therefore contains less tokens. For the reason that it was the first dataset to train the network on, it has been decided to go with the shortened version. The text file with the novel poses a certain inconvenience because it contains a lot of text decoration which can create noise in the model. But also it can have a positive effect on the user experience because it can be perceived as a unique style of the author. The unformatted version contains 26,413 words and 148,574 characters.

The second dataset used is the novel ‘Catcher in the Rye’ by J. D. Salinger (Valenzuela 2017). The version found on GitHub is slightly incomplete in the beginning and therefore was filled with the original text from the book. Also it was slightly formatted for the sake of the correctness of the output of the neural network. Chapter enumeration and the title page were removed from the text file. In the final revision there are 73,676 words and 382,008 characters.

The last and the longest dataset is the novel ‘Iliad’ by Homer (Stevenson 2000). It contains 152,507 words and 807,481 characters. It doesn’t contain much text decoration apart from chapter (book) enumeration. Also it has some translation references and copyright information. All this information was removed from the dataset since it could affect the training process. It still can be viewed in the original text document in the reference list. Another text decoration elements such as book (chapter) enumeration and dash symbols dividing the chapters was also removed.

Later as the project evolves more datasets potentially could be added. The main problem of having multiple datasets is the size of the checkpoint files created by the neural network for training purposes. ‘Alice in Wonderland’ checkpoint, which was trained 3,200 times on the novel is around 500MB and ‘Catcher in the Rye’ checkpoint is about the same size and the same amount of training steps. It would be possible to add more stories to the project if the checkpoint storage would be located not on the user’s machine but rather on a server.

Pillow

‘Pillow’ is a Python library used to manipulate images (Clark & Contributors 2022). It’s a fork from an original library named ‘PIL’. The reason why it was chosen to use in the development project is that it offers an easy and fast way to insert text into the image. This process is also known as ‘meme’ creation. It is possible to use generated samples from the neural network, either cut the size of the samples after it has been generated or generate a small sample in the first place. The second option reduces the time of generation process but it also takes away the option to format the text if the first and the last sentences are inappropriate. The customisation of the text in the library is a very straightforward process. It is possible to change the font, the size, the colour, the position and the angle. The ‘Meme mode’ section will most likely contain pictures in unified format (i.e. the font size, the colour and the angle will be the same).

In terms of the project, ‘Pillow’ will be used in the ‘Meme mode’ section. The idea is similar to the stories generator, the difference lies in the fact that the text is inserted into the image, rather than being printed below the image. Stories are presented in a post-like format.

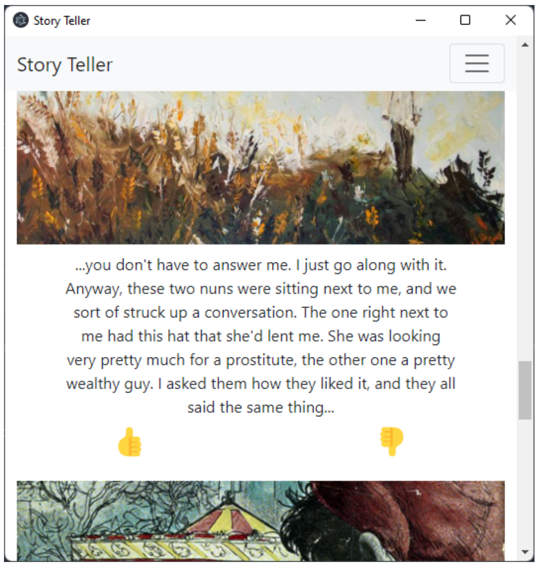
Frontend

The frontend of the project consists of a mixture of HTML, CSS and JavaScript. HTML is a markup language which is used to create the layout of the webpage. CSS is a language used to style the webpage. In this project it's used in a bundle with Bootstrap for easier manipulations with the styles. JavaScript is used to control the behaviour of the webpage and everything in it. Some of the functions displayed in the webpage are calculated and controlled solely with JavaScript with no interaction with Python.

Web Application appearence

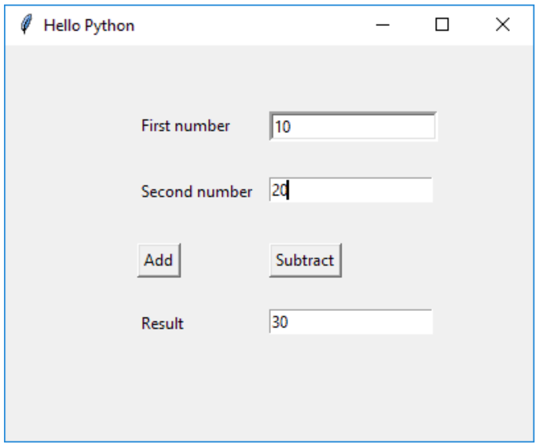
The application appearance is based on Bootstrap 5 and Electron. Bootstrap (Mark Otto & Contributors 2022) is a CSS framework/library that is used to create a responsive and mobile-first design. It's files (CSS and JS) can be accessed via 'Content Delivery Links' (CDN). This method is useful when only a part of the content hosted on CDN servers is used in development since CDN technology allows to deliver only necessary files. Also a big advantage is the loading time if the resource is server-based by reducing latency (CDNetworks 2021). The other aspect of the look is 'Electron' which was formerly known as 'Atom Shell' (Sawicki 2015). It's a cross-platform application that is used to develop application with web-like design. The main reason to work with 'Electron' is the ability to have the same look and feel on all platforms. GUI applications can be developed and deployed using web technologies much faster. Prior to the introduction of 'Electron' to the project, the application was developed on the basis of different browsers and operating systems.

Also as can be seen on the figures below, most popular approach to the development of Python GUI apps - Tkinter, doesn't satisfy the standards of app design in early 2020s.



...you don't have to answer me. I just go along with it. Anyway, these two nuns were sitting next to me, and we sort of struck up a conversation. The one right next to me had this hat that she'd lent me. She was looking very pretty much for a prostitute, the other one a pretty wealthy guy. I asked them how they liked it, and they all said the same thing...

Story Teller (Electron)

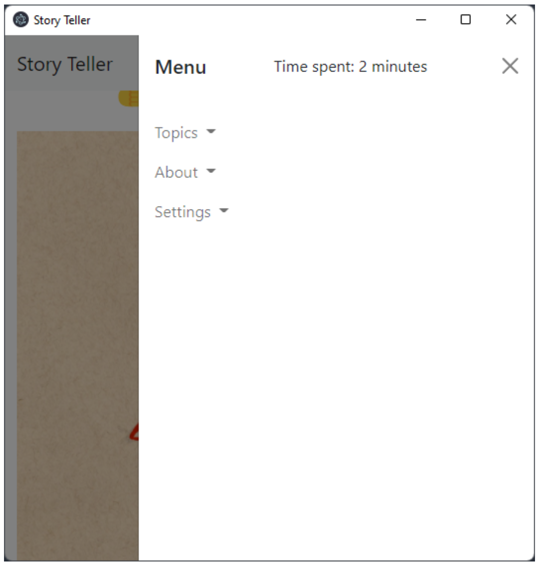


Tkinter app

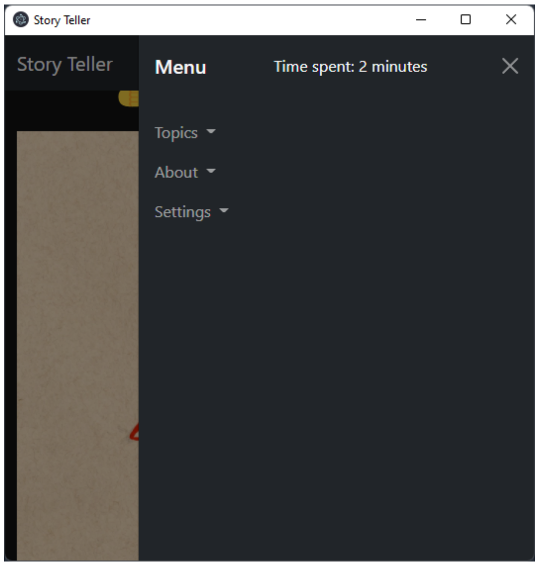
Figure 1: A comparison of two GUI layouts

Similar situation occurs when it comes to animation with Tkinter. Many examples and many guides were examined and it has been concluded that Tkinter, in spite of the fact that it's one of the most popular GUI libraries, is not suitable for the purpose of the project because the aim of the project is to create a fancy web-based application that can catch the attention of the user.

Another aspect of the design is the theme. Since many modern projects offer a possibility to change to either a light or a dark theme, it has been decided to develop an application with two themes. Bootstrap does most of the work for the theme change in terms of the colour change. Theme is persistent and is to a settings file which is stored in the application's folder. On every start-up the application checks if the theme is set to the light or dark mode accessing the JSON file.



Light theme



Dark theme

Figure 2: A comparison of two GUI themes

Displaying the content

To display the content it has been chosen to follow modern techniques of infinite scrolling. It loads the content continously while the content is being generated on the fly. Infinite scrolling absolutely matches the topic of the project since the main idea was to create stories without any administrator intervention and minimise any unnecessary clicks and interactions. The content is displayed in a post-like format - that means a random picture from a set is followed by a random and freshly generated text based on a neural network checkpoint. The displayed content depends on which topic is selected since the content is separated in different topics and displayed with different pictures.

Another aspect which was considered with exclusive scrupulousness is the conformity to accepted social standards. In view of the fact that neural networks do not understand major human problems of inequality, supremacism and just unacceptable language, a word filter can be used to filter out the most offensive words. A dictionary of such words can be formed and used to exclude the possible offensive language. Some content fed to the network might contain questionable statements and thus the output of the network will contain altered but still questionable words. The problem might be regarded from two perspectives: the first one is that it's not acceptable and the second one is that it's not even a problem because the output is generated based on the style of the original author.

User feedback

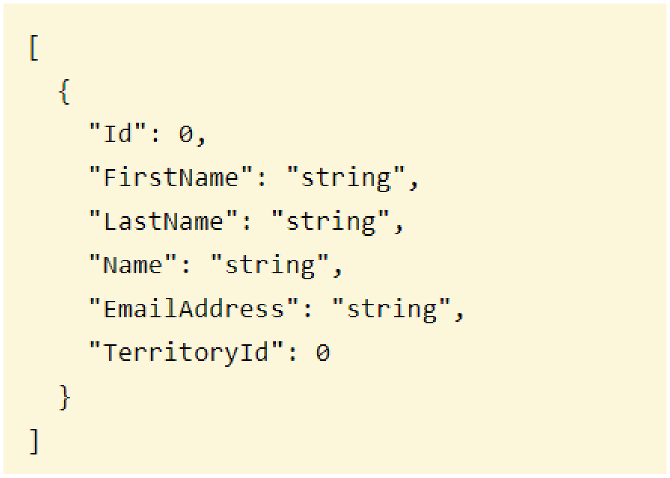
User feedback is an easy and effective way to improve the application and enhance the text generation process in particular. To establish a dialogue with a user the application uses a standard approach to the problem. The user can either like or dislike the text generated by the application by simply clicking one of the buttons. It is planned to store the reactions in the database and use them to manually assess the quality of the content. Comments are another possible solution to the user feedback problem but they seem too like a too complicated approach.

Database

Databases are used to store the user settings and the user feedback. The settings are stored in a JSON file and are accessed every time the application is started. JSON is a lightweight, human-readable text format that can be used with any programming language. The objects in the files are stored in an 'attribute-value' format.

The purpose of the traditional database is defeated since the application doesn't store any sensitive information. No encryption of the database is used either.

There are many other data serialisation formats such as XML, YAML, BSON etc. It has been decided to use JSON because of its simplicity and flexibility compared to YAML. YAML is a superset of JSON and it seemed to big for the purpose.



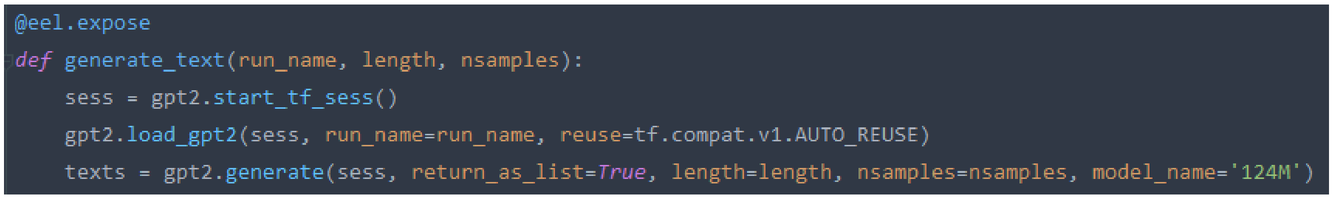
```
[
  {
    "Id": 0,
    "FirstName": "string",
    "LastName": "string",
    "Name": "string",
    "EmailAddress": "string",
    "TerritoryId": 0
  }
]
```

Figure 3: JSON database

Linking the frontend and backend

The main bridge between the frontend and backend used in this project is a 'little Python library' called 'Eel' (Knott 2022). It allows interaction between JavaScript and Python at every point of the development. The library is used to create a web server that can be accessed from the browser - Chrome, Safari, Firefox and last but not least - Electron.

The most important feature of 'Eel' is function exposition. It allows to call any function that is explicitly defined with a '@eel.expose' decorator and use it anywhere in the JavaScript code and vice versa.



```python
@eel.expose
def generate_text(run_name, length, nsamples):
    sess = gpt2.start_tf_sess()
    gpt2.load_gpt2(sess, run_name=run_name, reuse=tf.compat.v1.AUTO_REUSE)
    texts = gpt2.generate(sess, return_as_list=True, length=length, nsamples=nsamples, model_name='124M')
```

Figure 4: Eel exposition

It's important to note that when a function call is made from the JavaScript code, it should be done in async/await fashion. This is because the function call is made in the background and the result is returned after some time.

The separation of the frontend and backend is an important action becasue it allowed to use the most efficient approaches on each side and also focus on each part individually.

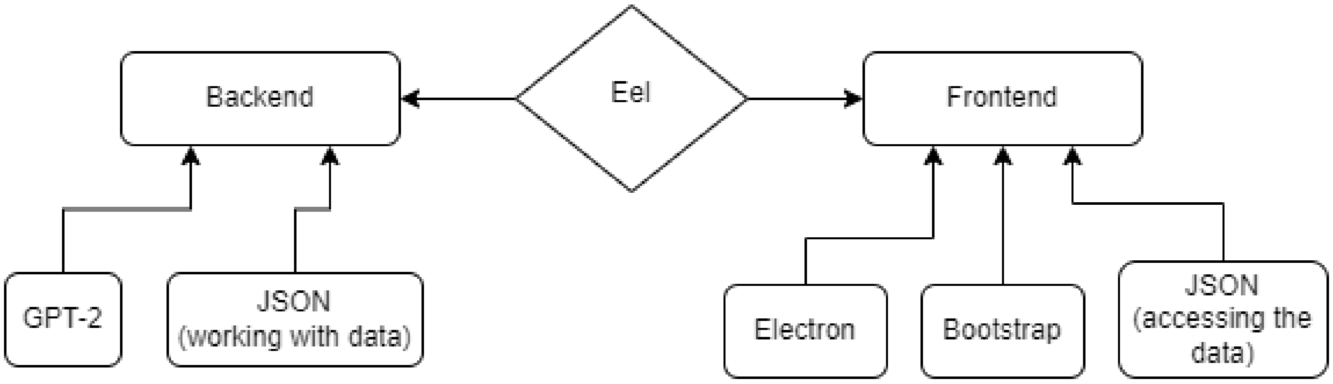


Figure 5: The relations in the system

Development lifecycle

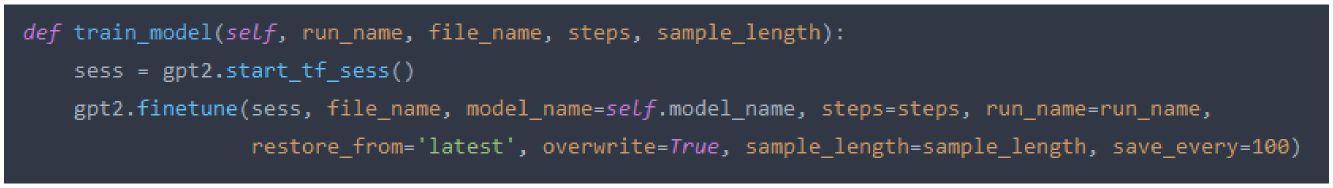
Development process followed the principles of Agile practice. The main aspects of the development methodology are: continiual improvement, flexible responses to any change and most importantly evolutionary development. Since the project is developed by a single person, not a team, some of the practices of agile methodology are not applicable.

Training the model

Implementation

Training or finetuning the model was a crucial step in the development process. Before the neural network would be able to generate specific text based on some dataset, it has to be trained on it. The training process involves tweaking many parameters of the model and the process is repeated until the best options are found. Currently, all three checkpoints are trained with the same parameters and on the same model - 124M (124 million parameters). The finetuning process begins with a creation of a new session. Then the the finetuning function itself is called with the following parameters: *dataset name, model name, number of steps, name of the run, restore point, overwrite option, length of the sample and an interval to save the checkpoints.*

Dataset name is the name of the preferred text file to train the network on. For this project three datasets were chosen - Alice, Catcher and Iliad. Model name is the name of the model used in the training process. It can be either '124M' or '355M'. The difference is in the parameter count. The '124M' model has 124 million parameters while the '355M' model has 355 million parameters. Number of steps is the parameter that determines how many times will the training process be repeated. For this project every checkpoint is trained more than 3,000 times in order to produce a meaningful result. Run name parameter specifies which checkpoint is being trained. It is useful to have multiple runs in the same model. Restore point and overwrite options are used for the sake of optimisation of the process to minimise the chance of failing (crashing). Also the interval to save the checkpoints is set to a small value of 100 to save the progress in case the process crashes.



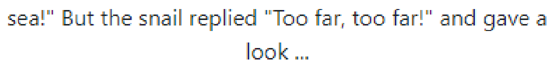
```
def train_model(self, run_name, file_name, steps, sample_length):
    sess = gpt2.start_tf_sess()
    gpt2.finetune(sess, file_name, model_name=self.model_name, steps=steps, run_name=run_name,
                  restore_from='latest', overwrite=True, sample_length=sample_length, save_every=100)
```

Figure 6: Training function

Testing

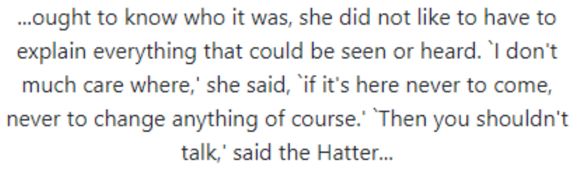
Testing the training process is not a very long process itself, so it is not described in detail. The 'GPT-2-Simple' library offers many parameters to tweak and test the performance of the training process. All datasets were fed to the network and trained on a machine with a 'NVIDIA RTX 3080' GPU - this allowed an effective and speedy training process.

Several ways of training the neural network on the first dataset 'Alice' were tried. The first attempt included the raw format of 'Alice in Wonderland' text file without any preprocessing (i.e. text is decorated with numerous spaces, unnecessary punctuation marks and different symbols). The results from the training were acceptable and the model was able to generate some of the samples that had sense in them. The second attempt included the preprocessing of the original text file. Chapter enumeration, excess symbols and other punctuation was reduced in order to minimise the impact on the network. Paragraph spaces were left in the text. The generated text was compared with the original text and the results were satisfactory.



sea!" But the snail replied "Too far, too far!" and gave a look ...

Unprocessed text (trained 13,100 steps)



...ought to know who it was, she did not like to have to explain everything that could be seen or heard. `I don't much care where,' she said, `if it's here never to come, never to change anything of course.' `Then you shouldn't talk,' said the Hatter...

Prepocessed text (trained 1,000 steps)

Figure 7: A comparison of unprocessed and processed datasets

The same process was applied to the second dataset 'Catcher in the Rye'. It was cleaned from the chapter enumeration and then fed to the neural network. It was trained more than 13,000 times.

The Iliad training was the fastest in terms of adjusting the text before feeding it. Since it didn't have any excessive decoration and the paragraphs didn't have indents - the output text from the network was of a very good quality.

The finetuning parameters weren't tweaked much. The learning rate stayed at 0.0001 the gradient count was set to 5. As for the steps, it's impossible to identify a universal value for the best result. The best result for each dataset was achieved by taking into account the accuracy of each training step. The longer and more complex the dataset - the longer it takes to train. The final accuracy the neural network aimed to achieve was *0.1-0.2*. Therefore training times for every dataset are very different.

Reflection

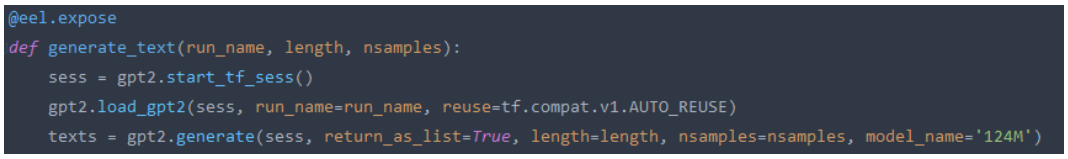
Training the model was one of the most time-consuming parts of the project. Possibly, another method of training could have been used. There's still a lot of freedom in what can be tweaked and changed in the parameter sections to enhance the results of the training. For example, the biggest enhancement would be to use a different model with more parameters. This project is limited computationally and therefore the model with the least parameters was chosen. The video memory of the machine wasn't enough to train the model with 355 million parameters. There is a solution to that problem as well (use memory saving gradients and only train transformer layers) but it's not implemented in Tensorflow 2.

Text generation

Text generation sprint could have been placed together with model training sprint. However, it was decided to separate the two because they are quite important on their own and whereas model training doesn't require frontend adaptation - text generation does because it is the final product that is being presented to the user.

Implementation

Text generation is the most important part of the project. The function takes a trained checkpoint of the network and based on the training it presents the user with a text. The function is written in a separate class based on the 'GPT-2-Simple' library. The function takes the following parameters: *run_name, length, nsamples*. These parameters are thought to be the most 'volatile' because they are changed often. The *'run_name'* parameter specifies which checkpoint is being used. The *'length'* parameter defines how many tokens the text should contain and the last parameter *'nsamples'* determines the number of samples to be generated.

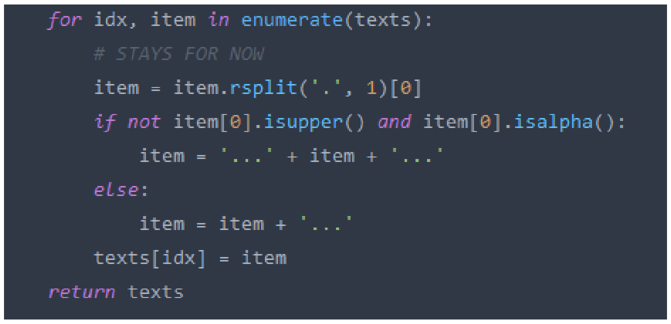


```
@eel.expose
def generate_text(run_name, length, nsamples):
    sess = gpt2.start_tf_sess()
    gpt2.load_gpt2(sess, run_name=run_name, reuse=tf.compat.v1.AUTO_REUSE)
    texts = gpt2.generate(sess, return_as_list=True, length=length, nsamples=nsamples, model_name='124M')
```

Figure 8: Eel exposed generate function

For the purpose of the infinite generation it is important to specify the *'reuse'* parameter when the GPT-2 is being loaded. This parameter allows the model to be reused without the need to load it again. In the example above this parameter is set to a value directly taken from the Tensorflow library since 'GPT-2-Simple' provided explanation in the documentation doesn't work (setting the 'reuse' parameter to 'True').

Yet this is only a first stage of the text processing in the function. After the text is generated it is handled by the algorithm that is shown on the below figure.



```
    for idx, item in enumerate(texts):
        # STAYS FOR NOW
        item = item.rsplit('.', 1)[0]
        if not item[0].isupper() and item[0].isalpha():
            item = '...' + item + '...'
        else:
            item = item + '...'
        texts[idx] = item
    return texts
```

Figure 9: Text processing algorithm

The presented algorithm is developed specifically for this project. It is based on the idea that the text which is output by the function is cut - it is been tested and proven in the previous section 'Model training'. The incomplete text seems to be cut only in the beginning and the end of the sample so, the algorithm iterates over the samples and removes the last sentence from each of them. This is done using Python 'rsplit' function which is the reverse version of a standard 'split'. After that, the first element of the word is checked for two conditions to decide where to put the ellipsis. First condition - uppercase, second condition - alphabetic character. If this is the case, the ellipsis is placed in the beginning and in the end of the word. If it's not - only in the end.

The point of the ellipsis is to make the text more pleasing to the eye and slightly confuse the perception of the user. It's been decided to call this trick 'Ellipsis of Uncertainty' - when the user sees the text that begins with ellipsis which signals that this is not the beginning of the text and ends with ellipsis which has the same meaning in perception,e tex they won't expect to see a fully understandable piece of text.

Testing

The testing process of the text generation involved the assessment of the generated text not only by the quality of the output but also by how it looks displayed on the page appended to the image. It might seem that this testing section overlaps with the testing section of the frontend development and it does, since the correlation between those elements is very high.

The main aspect of testing was the time taken to generate the text sample. It is important not to make the user wait for too long because it may result in a loss of the user's attention. The model testing appendix shows different test cases and results of parameter tweaking. The most optimal parameters were found to be: length of the text - *100*, number of samples - *6*. It satisfies the time requirement of 15 seconds for generating all 6 samples on whichever topic the user chooses. 'Alice' checkpoint has been tested more than the other checkpoints due to the fact that it's the first checkpoint that was developed. The results and knowledge gained from the testing are transferred from 'Alice' to other checkpoints successfully saving time.

Reflection

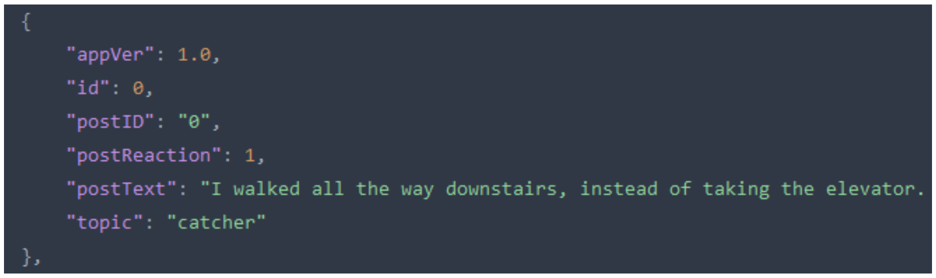
Output generation was a harder task to accomplish because it involved more than just using and adapting the functions from the library but also a 'Ellipsis of Uncertainty' trick had to be implemented. Also a lot of tests weren't passed because the expected outcome was an uncut text. That case wasn't reached and thus the trick described above was used. Besides, the text length was a question that had to be answered by testing what looks the best under a picture. Most likely, the reaction system will help to find out the best length and what exactly the user is looking for.

Reaction System

Implementation

Despite the simple explanation of the reaction system earlier in 'Major Components' section - the development itself was a bit more complex and involved the creation of a whole new logic system. Reactions now are used in a manual way by the developer to understand the problems of the system and make it more efficient and enjoyable to use. The idea is simple - if the user likes a generated post - they can react with a positive reaction (like). If they don't - they can react with a negative reaction (dislike). A logic system was needed in order to restrict the user from liking and disliking one post at the same time. It checks the boolean value of the reaction and if it's 'True', it's impossible to react with the opposite reaction again. Every reaction is separate for every post and has its own unique ID which is linked to the post. That way, it's easier to find the reactions in the database and understand a pattern of these reactions, since the main aim of the system is to gather collective information about the experience in whole and not one by one.

Apart from the logics - a database was implemented to store the reactions. JSON data serialisation format is used and every new reaction is stored there right away. Every object has several keys: *appVer*, *id*, *postID*, *postReaction*, *postTime* and *topic*.



```
{
    "appVer": 1.0,
    "id": 0,
    "postID": "0",
    "postReaction": 1,
    "postText": "I walked all the way downstairs, instead of taking the elevator.
    "topic": "catcher"
},
```

Figure 10: Example of an object

As every post can be reacted at, it can also be unreacted. This is an important feature which helps to tidy up the database in case if the user changes their mind or just missclicks. The system for unreacting is slightly more complex than reacting. It searches the database reversely and finds the last reaction of the user with a corresponding post ID. It must work from the end to the beginning because many entries might have the same post ID and only the last one should be deleted.

Testing

Reflection

The reaction system was one of the hardest things to implement since it has very convoluted logic. The main problem with it was the naming of the 'like' and 'dislike' buttons in the HTML document. After the issue has been solved, the implementation became rather simple. A better possible solution would be to use this feedback somehow to automatically improve the model and adjust the generated text without any manual intervention. This approach would make the app totally autonomous and would be a good point for the future development.

Bibliography

CDNetworks (2021), ‘Cdn vs local’.
URL: *https://www.cdnetworks.com/web-performance-blog/cdn-vs-local/*

Chen, T., Xu, B., Zhang, C. & Guestrin, C. (2016), ‘Training deep nets with sublinear memory cost’, *arXiv:1604.06174 [cs]* .
URL: *https://arxiv.org/abs/1604.06174*

Clark, A. & Contributors (2022), ‘Pillow’.
URL: *https://github.com/python-pillow/Pillow*

Johnsen, P. (2013), ‘Alice in wonderland’.
URL: *https://gist.github.com/phillipj/4944029*

Kay, A. (2007), ‘Tesseract: an open-source optical character recognition engine’.
Accessed Jan 8 2022 [Online].

Knott, C. (2022), ‘Eel’.
URL: *https://github.com/ChrisKnott/Eel*

Mark Otto, M. C. & Contributors (2022), ‘Bootstrap’.
URL: *https://github.com/twbs/bootstrap*

Musstafa (2022), ‘Optimizers in deep learning’.
URL: *https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019), ‘Language models are unsupervised multitask learners’.
Accessed Jan 6 2022 [Online].

Rosenblatt, F. (1958), ‘The perceptron: A probabilistic model for information storage and organization in the brain.’, *Psychological Review* **65**, 386–408.

Sawicki, K. (2015), ‘Atom shell is now electron — electron’.
URL: *https://www.electronjs.org/blog/electron*

Stevenson, D. C. (2000), ‘The iliad’.
URL: *http://classics.mit.edu/Homer/iliad.mb.txt*

Valenzuela, C. (2017), ‘the-catcher-rye’.
URL: *https://github.com/cvalenzuela/rwet/blob/master/final/source_text/ny/the-catcher-rye.txt*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L. & Polosukhin, I. (2017), ‘Attention is all you need’.
Accessed Jan 7 2022 [Online].

Woolf, M. (2021), ‘gpt-2-simple’.
URL: *https://github.com/minimaxir/gpt-2-simple*

Appendices

Testing

Model testing

Case	Input	Expected	Actual	Result
GEN0	Length: 10 Samples: 1 Batches: 1	No cut text Time: <15s	Text cut Time: 6.64s	Fail Pass
GEN1	Length: 1023 Samples: 1 Batches: 1	No cut text Time: <15s	Text cut Time: 12.99s	Fail Pass
GEN2	Length: 500 Samples: 6 Batches: 1	No cut text Time: <15s	Text cut Time: 30.48s	Fail Fail
GEN3	Length: 64 Samples: 2 Batches: 2	No cut text Time: <15s	Text cut Time: 5.41s	Fail Pass
GEN4	Length: 60 Samples: 2 Batches: 2	No cut text Time: <15s	Text cut Time: 5.33s	Fail Pass
GEN5	Length: 60 Samples: 2 Batches: 1	No cut text Time: <15s	Text cut Time: 5.43s	Fail Pass
GEN6	Length: 70 Samples: 2 Batches: 1	No cut text Time: <15s	Text cut Time: 5.63s	Fail Pass
GEN7	Length: 100 Samples: 6 Batches: 1	No cut text Time: <15s	Text cut Time: 9.46s	Fail Pass
GEN8	Length: 100 Samples: 6 Batches: 1	Time: <15s	Time: 12.12s	Pass
GEN9	Length: 100 Samples: 6 Batches: 1	Time: <15s	Time: 11.66s	Pass

Table 1: GEN0-GEN7 - ‘Alice’, GEN8-GEN9 - ‘Catcher’

Appendix B