Poem Generation Final Report

Jakub Bilski, Jakub Brojacz, Jakub Kosterna

May 2022

Contents

1 Introduction **2**

2 Code structure **3**
2.1 Main directory . . . 3
2.2 /src subdirectory . . . 3

3 Preparing data for training **4**

4 Detailed concept **5**
4.1 Accent model part . . . 6
4.1.1 SRC / TGT different values . . . 6
4.1.2 Embedding . . . 7
4.1.3 Transformer + Feedforward . . . 7
4.1.4 Accent tokenizer . . . 8
4.1.5 Transpoze . . . 9
4.2 GPT2 and its fine-tuning part . . . 10
4.2.1 GPT2 input 50000 different values . . . 10
4.2.2 GPT2 . . . 10
4.2.3 Remove first half (reference line) . . . 10
4.3 Combining both parts & final operations . . . 11
4.3.1 Elementwise multiplication . . . 11
4.3.2 log softmax . . . 12

5 External packages **13**

6 Results **14**
6.1 Model (accent prediction) after 100 epochs . . . 14
6.2 Gpt2 fine-tuning . . . 15
6.2.1 before fine-tuning . . . 15
6.2.2 after fine-tuning . . . 16
6.3 (Model+fine-tuned gpt2) vs (fine-tuned gpt2) . . . 17

7 Summary **19**

1 Introduction

The project task was to design and implement a poetry generating algorithm. We limited the problem to texts in English, performing artificial poetry generation based on a fine-tuned model along with our additional solution taking into account word accents.

The core of the solution is the pretrained GPT2 system for generating poetry. This approach outputs a word based on the given previous words in the input text, instead of a line to line architecture, which might seem like the most natural solution.

In addition to using the pretrained solution, we also fine-tune it based on additional data preprocessed by us. Then we also include a model that predicts the next words based on the accents found automatically. In fact, the whole point of the project was to solve a problem: how to reconcile this pretrained GPT2 solution with our proprietary model that made a more "thoughtful" prediction based on the accents found.

The operation of the main part of the code, after preparing the input data, is shown in figure 2. In other files in src/ directory we implemented our own architecture using GTP2 fine-tuned net with additional transformer responsible for the rhythm of the poem.

2 Code structure

The complete solution is available on the project's GitHub page. The model and files concerning the scheme 2 are located in the */src* subfolder, while the other files are located in the root directory.

Most of the files from the main level were prepared before programming the actual part in the /src folder and were required for its execution.

2.1 Main directory

At this level, we can see five individual files of various types, as well as one folder. The following files are responsible for:

- **src** subdirectory - is the core of the model, to which the following subsection is devoted.
- **PREDICTIONS** Markdown file - contains some examples of model usage and results. These will also be cited in this report.
- **Poem_Generation_Stage1** .pdf file - here is a report after first milestone
- **README** markdown file - key info about the project
- **data-processing** Jupyter Notebook script - here is a recreation of poetry from Kaggle Michael Aram's Poems Dataset (NLP), as well as their appropriate processing towards use by the correct model.
- **google-20000-english** text data file - contains a list of the 20 000 most common English words in order of frequency, as determined by n-gram frequency analysis of the Google's Trillion Word Corpus. Source: Josh Kaufman's google-10000-english GitHub repository
- **verses_pairs** .csv data file - created 87265 pairs of verses for GPT2 model fine tuning operation, created by *data_processing* Jupyter Notebook file.

2.2 /src subdirectory

This subfolder contains nine scripts that perform operations from figure 2. The main file containing the solution implementation is *main_for_eden.py*. It contains the core of the operation and references to other scripts that are responsible for minor operations. The logic behind these nine scripts is explained in Chapter 4, Detailed concept.

The way the files are connected and reference each other is presented in figure 1. An arrow from block A to block B means that in the script corresponding to block B, the code pertaining to block A is imported.

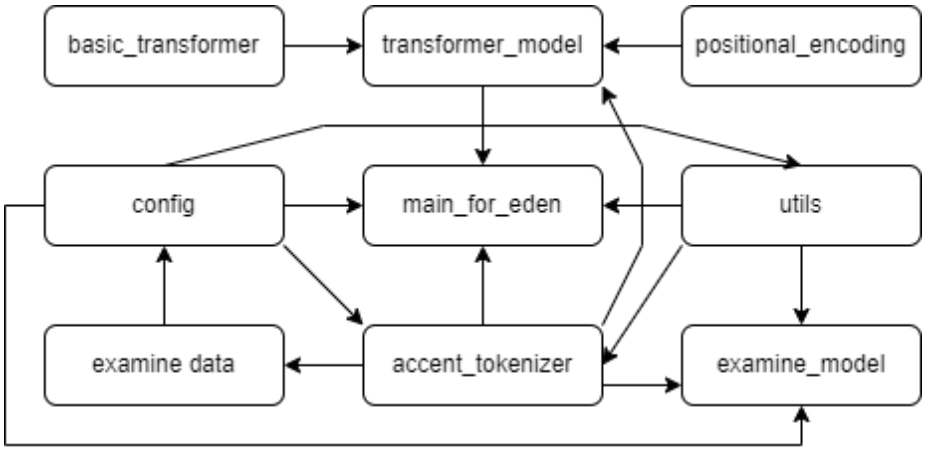

Figure 1: /src files references graph

3 Preparing data for training

GPT2 is not a rigid solution taken from the Internet - it is additionally fine-tuned with data from Kaggle Michael Aram's Poems Dataset (NLP), after appropriate processing. The processing script is *data-processing.ipynb* and includes among others:

1. loading over 20,000 poems from two folders and 279 subdirectories to a list
2. removal of non-English poetry, using *langdetect* package
3. changing all letters to lower case and disregarding punctuation
4. deleting excessively short or long lines
5. disregarding selected types of unusual poetry (eg. poems based on single characters or deleting lines containing "copyright" word)
6. removing poems containing words that are rare in the rest of the dataset
7. leaving only lines with words that are in the dictionary of most frequently used words according to the external collection (using file from Josh Kaufman's google-10000-english GitHub repository

The 87265 so created at the end are saved in the *verses_pairs* CSV file.

4 Detailed concept

In order to present our solution, let's analyze the following flowchart blocks from the figure 2

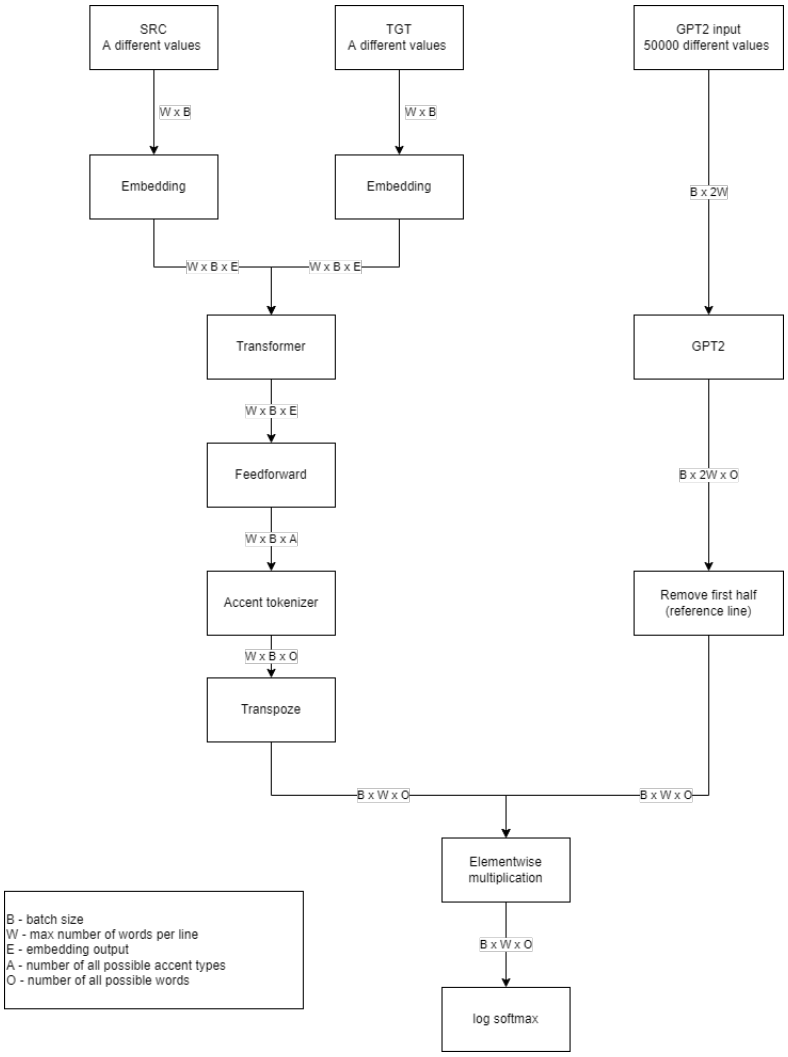

Figure 2: Flowchart of the presented model

In the first stage, 87 265 pairs of verses from *verses_pairs.csv* file described in the previous section, are loaded. These will be used in both parts of model, which is a combination of two parts: 1) building a transformer that takes accents into account [section 4.1] and 2) fine-tuning GPT2 [section 4.2]. In the end, both submodels will be connected [section 4.3.

Two tokenizers are needed to create the input:

1. needed for GPT2 input:

   ```
   t_tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
   ```

   ... which is taken strictly from the library,

2. for the input of our accent model:

   ```
   a_tokenizer = accent_tokenizer.AccentTokenizer.create(
       t_tokenizer, config.ROOT_PATH / "tmp.pickle")
   ```

   ... refering to the *AccentTokenizer* class and created pickle file.

Then the input data is created:

```
utils.create_input_data(pair[0], pair[1], t_tokenizer, a_tokenizer)
```

... referencing function from *utils.py*, performing tokenizing and padding of input data. It provides a "predicting" line and a "predicted" line by the model, as well as two tokenizers. The output of this function is: first the tokenized first two lines of our model, then the input to GPT2 and the information about the number of tokens returned by GPT2 in the first line.

It is important to note that both post tokenization and accent tokenizer data are saved in the file, which allow the program to run faster.

4.1 Accent model part

4.1.1 SRC / TGT different values

The transformer architecture is built of two components:

- **src** - the input to the encoder
- **tgt** - the input to the decoder

Firstly, zeros masks for src and lower triangular matrix for tgt are created. Their size is imposed by MAX_LENGTH parameter from *config.py*. By default, it takes the value of 20, which corresponds to a fixed number of ten words. This limit is required, because of properties of a vector to vector model. In case of a line shorter than the assumed length, a padding token is made. It is important that the function that counts the loss and trains the network ignores the fragments with padding.

Then, given matrixes are filled with values from manually prepaired input data. src gets first elements from pairs (first line) and tgt - second ones (second line). If a number of words is less or equal to MAX_LENGTH parameter, the words are completed and the rest of elements in a row stays the same; otherwise, as many words as the limit are filled in.

4.1.2 Embedding

In order to perform operations on the transformer, number vectors are needed instead of lists of words.

GPT2 operates on tokens, and in our case tokens are not always exactly words. Some words are left whole and some are split in half. For example, the word *chocolate* is split into two parts: *ch* and *ocolate*. The solution was prepared with the help of embeddings, which are created based on the dictionary of words that are in the input.

Once the matrix is prepared, at the very end we use it to multiply the accents to get the word chances, which in turn is multiplied by the word chances returned by GPT2. Additionally, normalization is applied so that instead of the matrix being divided by ones, we normalize the chance per accent (for example, if 200 words have a given accent, each gets a value of not one, but 1/200).

The operation is performed for both src and tgt matrixes, analogously. This results in a pair src x tgt for which, for each being a W x B matrix of embeddings, where W - MAX_LENGTH and B - number of input rows. Used embeddings are pretrained, we don't create them manually.

4.1.3 Transformer + Feedforward

Having ready src and tgt structures, the transformer can be built. The operations are performed in the *transformer_model.py* script.

The transformer-related parameters that determine the operation of our model are described in *main_for_eden.py* file:

- d_model = 200 - the number of expected features in the encoder/decoder inputs. In our case there are word embeddings.
- nhead = 2 - number of heads in the encoder/decoder of the transformer model
- nlayers = 2 - number of layers
- dropout = 0.2 - probability of element to randomly become zero on every forward call

The network is learned to find and predict accents in the given texts based on src and tgt. The transformer's role is to do it. In order to achieve is, the feed-forward layer implemented. Its weights are trained and the exact same matrix is applied to each respective token position. The input dataset is split into train set and test set in a 4:1 ratio.

Its role and the purpose is to process the output from one attention layer in such a way that it better matches the input for the next attention layer.

In addition, learning rates can also be adapted manually - the constants *MODEL_LR* and *GPT2_LR* in the *config.py* are responsible for this.

The result of this stage is similar in comparison of the previous one, but instead of embeddings, the number of all possible accent types is received.

4.1.4 Accent tokenizer

The word that is received at the very beginning, for the purpose of our part is tokenized using *accent_tokenizer.py* script. This operation is based, among others, on an external library *isletool*. Training is based on the cost function that we minimize to the word that really occurred.

Here (figure 3), the result of extra script *examine_data.py*, which shows the accent distribution in the dataset:

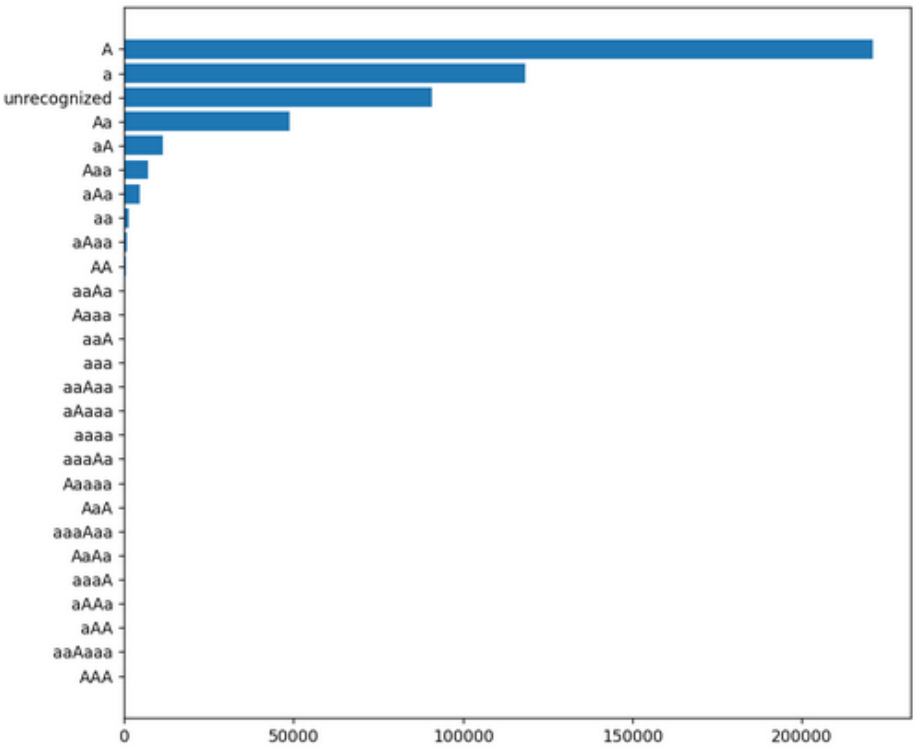

Figure 3: Model (accent prediction) after 1 epoch:

From this visualization, we can see that in the absolute majority of cases words with accents are generated: A, a, Aa, aA, Aaa and aAa. Others occur much less frequently.

The method from this library returns a word divided into syllables, along with marking the syllable with accent. Such an accent prediction model after 100 epochs gives the result (A - accented syllable, a - unaccented syllable):

- For input first line: Once upon a midnight dreary

  Accents detected in first: a aA a Aa A A Aa no_accent no_accent

- For input part of second line: while I pondered

Accents detected in second: a A A no_accent no_accent

Then, candidates for the next word in terms of accent are calculated. For the case above, the best candidates are:

1. A: 99.43318367004395
2. a: 0.5643955431878567
3. Aa: 0.0022517751858686097
4. : 0.00016407100247306516
5. aA: 6.774691918565168e-10

Following this operation, the next word is added according to the accent drawn - higher probabilities will naturally have words with accents having a higher percentage, while after that it still depends on the temperature parameter, which will still be quoted.

It may happen that a lack of accent is predicted for the next word. Indeed, the used found solution contains words in which no unambiguous accentuation was detected. However, the number of words without a found accent is not significant and does not significantly affect the solution - see the figure 3.

4.1.5 Transpoze

In the *PoemGenerator* class, the array is created with the help of *AccentTokenizer*, which is implemented in the *AccentToGtp2Tokenizer* class from the auxiliary *accent_tokenizer.py* script. Specifically: for each accent, all words so accented are retrieved and the resulting matrix is filled with ones where that accent corresponds to that word.

The matrix is needed for the next stages, in order to connect accent submodel with fine-tuned GPU2. It is created in *accent_tokenizer.py* file

```
matrix = np.zeros((t_tokenizer.num_words, self.num_words))
        for gtp2_id in tqdm(t_tokenizer.decoder.keys(), \
              desc="Initialising AccentTokenizer part 2"):
            for word, ids in word_to_gtp2_ids.items():
                if gtp2_id in ids:
                    accent = word_to_accent_id.get(
                        word, self.no_accent_id)
                    matrix[gtp2_id, accent] = 1
            if np.sum(matrix[gtp2_id, :]) == 0:
                matrix[gtp2_id, :] += 1
        matrix /= np.sum(matrix, axis=0)
```

For each accent, all words that are accented in the same way are taken. The matrix is filled with ones where the accent corresponds to that word. Note,

however, that GPT2 operates on tokens - so the text is converted to its internal representation.

The final result is therefore the chances of individual accents for the input data.

4.2 GPT2 and its fine-tuning part

4.2.1 GPT2 input 50000 different values

Generative Pre-trained Transformer 2 (GPT-2) is an open-source artificial intelligence created by OpenAI in February 2019. In other to load the core, which we will then fine-tune, we use pre-trained model from *transformer.GPT2Tokenizer* package. Just loading the solution takes one line.

4.2.2 GPT2

After the transformer is learned, the next syllables are used in conjunction with GPT2 to predict the next words of a line. The program can be reduced to predicting lines, but really at each stage it is all about predicting the next words anyway.

It's worth mentioning that this fine tuning is done completely separately from our module - theoretically they could be trained after the merger, i.e. after the predictions of one and the other network are multiplied by each other and trained only after that... whereas in our vision we are tuning GPT2 separately, making it - in familiar language - more "poetic". Only after one is trained - the other is trained and we make the connection of models (Elementwise Multiplication block) using multiplication.

4.2.3 Remove first half (reference line)

Main genesis involves the assumption, each line is generated only from the previous line. This allows a line to be generated "indefinitely". But when do we know to stop and move on to the next one?

No end tokens are created in the algorithm. In fact a line can be cut at any time. The implementation of end tokens would be difficult given the connection to the GPT2 solution, which does not return end tokens. In this variant, words are generated until explicitly told to stop adding words... and the same approach is used here. Unfortunately, it we didn't implement a good way to check if a word is appropriate at the end of a line. However, this could be done at least naively by checking how often a word was the last word on a line among input pairs.

The final result is therefore the chances of individual words for the input data.

4.3 Combining both parts & final operations

4.3.1 Elementwise multiplication

The final step is to put together an accent transformer along with a fine-tuned GPT2, having the chances of individual accents from the first and the chances of individual words in the second. In order to do it, the new N x M matrix is generated, where N - the number of accents, M - the number of output words from GTPT2. The whole operation is performed in the *test_model_and_gpt2_on_line* function in the main script, in the *get_matrix()* method of the model:

```
model_output = model.get_matrix(
            src, tgt_input,
            src_mask, tgt_mask,
            src_padding_mask, tgt_padding_mask,
            src_padding_mask)
        model_output = model_output[next_line_id].squeeze()
```

The full model is available in the *transformer_model.py* script and has two modes:

1. forward - "regular", returning chances for accents
2. get_matrix - improved forward - returns chances for words, based on accents

After chances for words have been received (one probability times the other probability), as a result the final chances after correction are obtained. In this case, no longer the operation of multiplication, but addition of logarithms is performed.

The end result can be seen in the example:
First line: when you come round a corner and

- High temperature:
    1. With model: ive no time to stop it my friends will id be as bold as you can be as you
    2. No model: to be acknowledged never a one has a word is spoken to you she is no more brave for
- Low temperature:
    1. With model: ive no doubt id out just be dead for all my fun shell be not my hero i is
    2. No model: for to have a ball between them they have no choice but to have him in them it a

4.3.2 log softmax

As is shown above, one more parameter is important - the values of low temperature and high temperature correspond to the numbers 0.1 and 0.8, respectively. The parameter is used in:

```
output = F.softmax(output.div(temp), dim=-1).cpu()
```

... in the *main_for_eden* file. When using softmax() in this way, for low values of temperature, the word with the highest chance is almost always chosen; as the value increases, there is more randomness. These numbers were chosen by intuition, and other values in the range (0; 1) could be tested longer. However, it is difficult to indicate the best number here or compare them objectively - we are still dealing with artificial generation of new text, which is difficult to evaluate for correctness only automatically. On the plus side, however, this argument is set after the model has been trained, which speeds up the process.

5 External packages

In addition to the code written from scratch, we used several additional libraries that helped us to prove the solution in an accessible way.

- **torch 1.11** - accent_tokenizer, basic_transformer, config, main_for_eden, positional_encoding, transformer_model, utils
- **pathlib 1.0.1** - accent_tokenizer, data_processing, config
- **tqdm 4.64.0** - accent_tokenizer, basic_transformer, main_for_eden
- **__future__ 0.18.2** - basic_transformer, main_for_eden
- **pandas 1.4.2** - data_processing, main_for_eden
- **pysle.isletool 4.0.0** - accent_tokenizer, main_for_eden
- **langdetect.detect 1.0.9** - data_processing
- **numpy 1.22.4** - accent_tokenizer
- **transformers.GPT2Tokenizer 4.19.2** - main_for_eden
- **typing 3.10.0.0** - basic_transformer
- **math** - basic_transformer, positional_encoding
- **time** - basic_transformer, main_for_eden
- **collections** - basic_transformer
- **io.open** - basic_transformer
- **pickle** - accent_tokenizer
- **random.sample** - data_processing
- **re** - basic_transformer
- **string.ascii_letters** - data_processing
- **sys** - main_for_eden
- **unicodedata** - basic_transformer
- **zipfile** - data_processing.ipynb

6 Results

The testing is done in *test_model_and_gpt2_on_line* in *main_for_eden.py* file. The key parameter there is *zero_line_text*, which is responsible for the first line.

6.1 Model (accent prediction) after 100 epochs

First, let us cite three cases of accent prediction based on the first line and part of the second line.

Example A

- For input first line: once upon a midnight dreary
  Accents detected in first: a aA a Aa A A Aa no_accent no_accent
- For input part of second line: while I pondered
  Accents detected in second: a A A no_accent no_accent

Best candidates for next word:

1. A: 99.43318367004395
2. a: 0.5643955431878567
3. Aa: 0.0022517751858686097
4. : 0.00016407100247306516
5. aA: 6.774691918565168e-10

Example B

- For input first line: Stasis in darkness. Then the substanceless blue
  Accents detected in first: A a Aa A a Aa A no_accent no_accent ...
- For input part of second line: Pour of tor and distances.
  Accents detected in second: A a a A Aaa no_accent no_accent ...

Best candidates for next word:

1. : 83.28053951263428
2. A: 16.673557460308075
3. a: 0.04590644966810942
4. Aa: 9.097684383618798e-07
5. aA: 1.4540406097045266e-11

Example C

- For input first line: Do not go gentle into that good night,
  Accents detected in first: A A A Aa aa A a A no_accent no_accent...
- For input part of second line: Old age should burn
  Accents detected in second: A A A no_accent no_accent no_accent ...

Best candidates for next word:

1. A: 99.9380350112915
2. a: 0.06134043214842677
3. : 0.0005922586296946974
4. Aa: 2.255896447422856e-05
5. aA: aA: 1.9488731375222335e-10

Lines are padded to 20 tokens, so ... means a lot of no_accent because of the padding.

6.2 Gpt2 fine-tuning

6.2.1 before fine-tuning

Example A

- Input first line: Once upon a midnight dreary
- Input part of second line: while I pondered
- High temperature: while I pondered over the details of my past, my heart began to throb and I felt my life getting closer
- Low temperature: while I pondered the fate of the world, I was struck by the sudden and unexpected change in the world.

Example B

- Input first line: Stasis in darkness. Then the substanceless blue
- Input part of second line: Pour of tor and distances.
- High temperature: Pour of tor and distances. "Let us go. This is the place. The whole place. It is
- Low temperature: Pour of tor and distances. Then the substanceless blue Pour of tor and distances. Then the substanceless blue Pour of tor

Example C

- Input first line: Do not go gentle into that good night,
- Input part of second line: Old age should burn
- High temperature: Old age should burn you." The sun is not set. It is only at sunrise, when the
- Low temperature: Old age should burn you to the ground. "I will not go to sleep until you have been

6.2.2 after fine-tuning

Example A

- Input first line: Once upon a midnight dreary
- Input part of second line: while I pondered
- High temperature: while I pondered on a golden news report on scary sea whose flesh and on your hand for grief whom out laughter
- Low temperature: while I pondered gives again all to me and any rather burn fire generous nest of gold and praise thee and on

Example B

- Input first line: Stasis in darkness. Then the substanceless blue
- Input part of second line: Pour of tor and distances.
- High temperature: Pour of tor and distances. by air through thick count brit field though their soft and by way so long enough for a
- Low temperature: Pour of tor and distances. by a single count or a single finger crime or heaven you may and by your ways to me

Example C

- Input first line: Do not go gentle into that good night,
- Input part of second line: Old age should burn
- High temperature: Old age should burn up todoes something extraordinary all that touches him with their own thought room is most ones heart shaped
- Low temperature: Old age should burn up that first time for you to try to eat them very smile and dear lord and do not

6.3 (Model+fine-tuned gpt2) vs (fine-tuned gpt2)

Example A First line: when you come round a corner and

- High temperature:
  1. With model: ive no time to stop it my friends will id be as bold as you can be as you
  2. No model: to be acknowledged never a one has a word is spoken to you she is no more brave for
- Low temperature:
  1. With model: ive no doubt id out just be dead for all my fun shell be not my hero i is
  2. No model: for to have a ball between them they have no choice but to have him in them it a

Example B First line: centuries ago some built fires in caves of stone

- High temperature:
  1. With model: ive found a world of dread ive by ur side a british or rome or
  2. No model: bloom in grown fat gown and multi games with various teaching skill in their kids gown and stamps out
- Low temperature:
  1. With model: ive found a free form of space for us in an infinite unique rome our an out right
  2. No model: bloom in rage of grief and sorrow sin i cried for pain far calm memory for grief and grief

Example C First line: cool to look to something outside

- High temperature:
  1. With model: ive meet my odd odd odd hair may be found atans part in my heart so now i
  2. No model: yourself something else very wrong something greater something else will happen that i seem bad to be in hot
- Low temperature:
  1. With model: ive meetin you in heaven all i know is i in heaven all my bread is so deep
  2. No model: yourself something else to abuse something to set you straight down that deeds and you will not see again

Example D First line: as large as a tank aiming

- High temperature:
    1. With model: ive notfound in that world of art work in sight of their own world for us we care
    2. No model: whether to leave whether truth or substance failed so please give us bear it forever perhaps some say it
- Low temperature:
    1. With model: ive run out of time i am young as age men are of old we know now be a
    2. No model: at ride its not a story that we all paint of him as a memory for him to be

Comparing the versions with fine-tuned GPT2 alone along with the variant with our model on accents combined, after taking the model into account (multiplying the odds by the accents), the result is more single-syllable words, although it is hard to say objectively whether they are certainly more accurate.

7 Summary

During the last couple of months, out team implemented a program to generate poetry. Successively we managed to train the GPT2 model with the data we found and processed, and to complete it with a solution that takes accents into account. Different versions of the program (with or without the previous steps) give different results and can be adjusted differently - for example, by changing the temperature parameter. Unfortunately, it is hard to apply an objective evaluation of the results and find a measure which will compare the quality of the rows, even for the real, original ones. The solution can be developed in the future by adding a system that captures the last words in the poem or including punctuation in the output.