HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Computer Science department

Málvinnsla

T-725-MALV

Assignment 1

Saldana Giorgio
Email: giorgio24@ru.is

10th October 2024

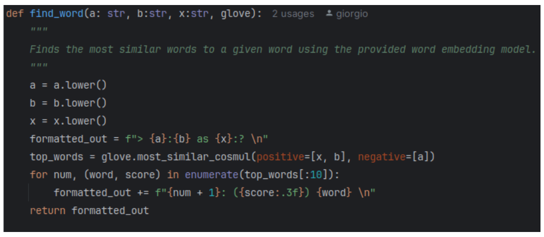
Contents

1 Bias in Word Embeddings

In this section, the answers to the questions posed in task 3 called "Bias in Word Embeddings" present in Assignment 1 of the Málvinnsla course will be presented. For reproducibility of the results, is possible to find the code in the public github repository at the following link here.

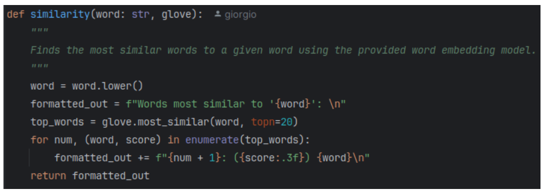
What kinds of biases are present in either dataset, if any?

To answer this initial question, a python code called word_bias.py was developed that allows you to extract similarities from words and find the most similar words using a particular proportion of the type a:b as x=?. The development to extract these features was done through these 2 functions built on top of the glove package imported at the beginning of the code.



```python
def find_word(a: str, b:str, x:str, glove):
    """
    Finds the most similar words to a given word using the provided word embedding model.
    """
    a = a.lower()
    b = b.lower()
    x = x.lower()
    formatted_out = f"> {a}:{b} as {x}:? \n"
    top_words = glove.most_similar_cosmul(positive=[x, b], negative=[a])
    for num, (word, score) in enumerate(top_words[:10]):
        formatted_out += f"{num + 1}: ({score:.3f}) {word} \n"
    return formatted_out
```

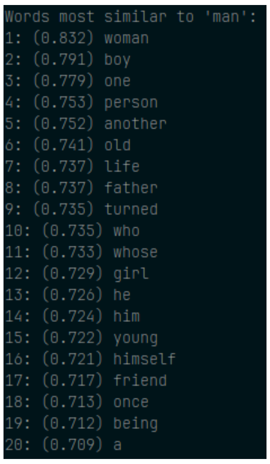
Figure 1: find_word function



```python
def similarity(word: str, glove):
    """
    Finds the most similar words to a given word using the provided word embedding model.
    """
    word = word.lower()
    formatted_out = f"Words most similar to '{word}': \n"
    top_words = glove.most_similar(word, topn=20)
    for num, (word, score) in enumerate(top_words):
        formatted_out += f"{num + 1}: ({score:.3f}) {word}\n"
    return formatted_out
```

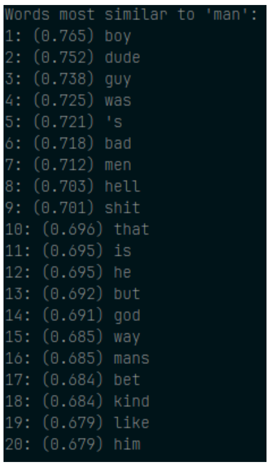
Figure 2: similarity function

These functions have been mostly taken from lab 3 and adapted for better formatted output, below is some of the output obtained from these functions.



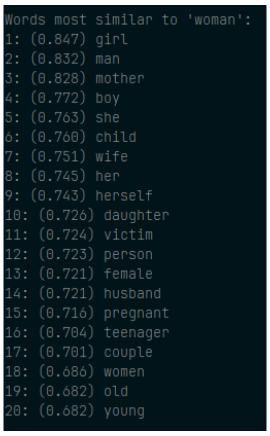
```
Words most similar to 'man':
1: (0.832) woman
2: (0.791) boy
3: (0.779) one
4: (0.753) person
5: (0.752) another
6: (0.741) old
7: (0.737) life
8: (0.737) father
9: (0.735) turned
10: (0.735) who
11: (0.733) whose
12: (0.729) girl
13: (0.726) he
14: (0.724) him
15: (0.722) young
16: (0.721) himself
17: (0.717) friend
18: (0.713) once
19: (0.712) being
20: (0.709) a
```

Figure 3: Similar words to "man" - Wikipedia dataset



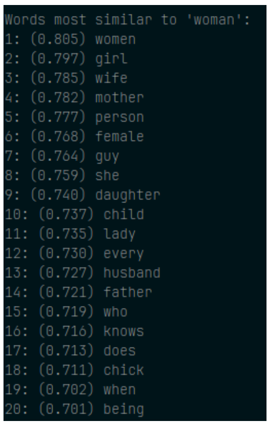
```
Words most similar to 'man':
1: (0.765) boy
2: (0.752) dude
3: (0.738) guy
4: (0.725) was
5: (0.721) 's
6: (0.718) bad
7: (0.712) men
8: (0.703) hell
9: (0.701) shit
10: (0.696) that
11: (0.695) is
12: (0.695) he
13: (0.692) but
14: (0.691) god
15: (0.685) way
16: (0.685) mans
17: (0.684) bet
18: (0.684) kind
19: (0.679) like
20: (0.679) him
```

Figure 4: Similar words to "man" - Twitter dataset



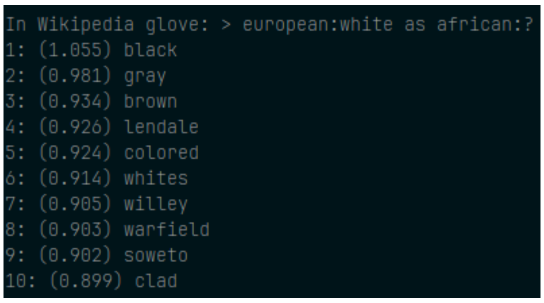
```
Words most similar to 'woman':
1: (0.847) girl
2: (0.832) man
3: (0.828) mother
4: (0.772) boy
5: (0.763) she
6: (0.760) child
7: (0.751) wife
8: (0.745) her
9: (0.743) herself
10: (0.726) daughter
11: (0.724) victim
12: (0.723) person
13: (0.721) female
14: (0.721) husband
15: (0.716) pregnant
16: (0.704) teenager
17: (0.701) couple
18: (0.686) women
19: (0.682) old
20: (0.682) young
```

Figure 5: Similar words to "woman" - Wikipedia dataset



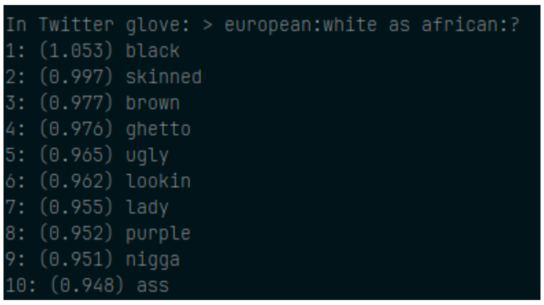
```
Words most similar to 'woman':
1: (0.805) women
2: (0.797) girl
3: (0.785) wife
4: (0.782) mother
5: (0.777) person
6: (0.768) female
7: (0.764) guy
8: (0.759) she
9: (0.740) daughter
10: (0.737) child
11: (0.735) lady
12: (0.730) every
13: (0.727) husband
14: (0.721) father
15: (0.719) who
16: (0.716) knows
17: (0.713) does
18: (0.711) chick
19: (0.702) when
20: (0.701) being
```

Figure 6: Similar words to "woman" - Twitter dataset



```
In Wikipedia glove: > european:white as african:?
1: (1.055) black
2: (0.981) gray
3: (0.934) brown
4: (0.926) lendale
5: (0.924) colored
6: (0.914) whites
7: (0.905) willey
8: (0.903) warfield
9: (0.902) soweto
10: (0.899) clad
```

Figure 7: Similarity for Racism Wiki



```
In Twitter glove: > european:white as african:?
1: (1.053) black
2: (0.997) skinned
3: (0.977) brown
4: (0.976) ghetto
5: (0.965) ugly
6: (0.962) lookin
7: (0.955) lady
8: (0.952) purple
9: (0.951) nigga
10: (0.948) ass
```

Figure 8: Similarity for Racism Twitter

Based on these outputs, various forms of bias can be identified in both Wikipedia-based GloVe embeddings and Twitter-based GloVe embeddings like:

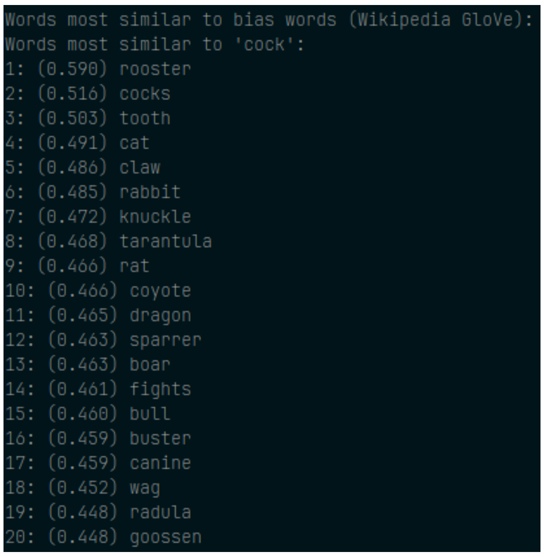
- **Gender Bias**: In both Wikipedia and Twitter embeddings, "man" is associated with terms like "mechanic," and "woman" is linked with occupations such as "hairdresser" and "housekeeper." This reflects stereotypical gender roles in society, where certain professions are associated with men and others with women. Additionally in the Twitter dataset , "man" is associated with words like "dude" and "guy," while "woman" is linked to "wife" and "mother," reinforcing traditional gender roles.
- **Ethnic Bias**: The comparison of "European" with "African" shows that words like "black" and terms with negative connotations (e.g., "ghetto," "ugly") are associated with "African," especially in the Twitter dataset. This indicates a racial bias in the way certain ethnicities are represented in the data.
- **Sexual Orientation Bias**: The terms "gay" and "lesbian" in both datasets have strong associations with words like "homosexual," "gays," and "bisexual," but in the Twitter dataset, the associations extend to derogatory terms like "fake," "homo," and "slut." This indicates a bias that negatively stereotypes LGBTQ+ individuals in the Twitter embeddings.

Are there any differences with respect to bias between the two kinds of data?

Yes, there are noticeable differences in bias between the Wikipedia and Twitter embeddings:

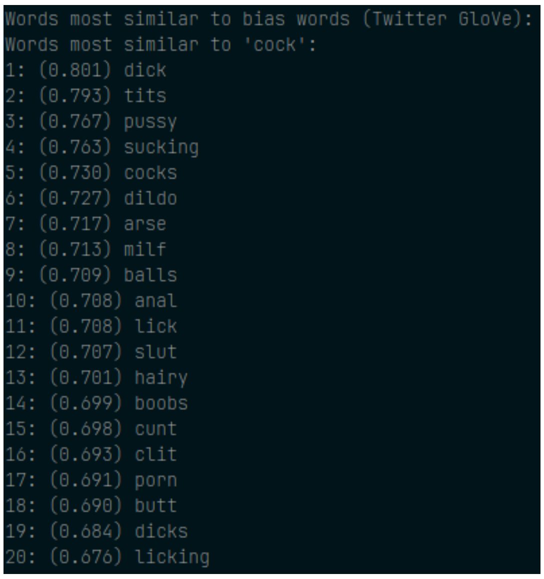
- **Wikipedia GloVe embeddings** seem to have more neutral or academic associations. For example, "woman" is linked to words like "person" and "female," while terms like "African" are more likely to be associated with geographic or cultural terms like "continent" or "nation." However, gender and ethnic stereotypes still persist, albeit in a more subtle form.
- **Twitter GloVe embeddings** have much more explicit and often derogatory associations. For example, terms related to gender and sexuality ("tits," "lesbian") are highly sexualized and associated with offensive slang. This reflects the informal, conversational, and sometimes offensive nature of Twitter as a platform.

Another example of difference between bias is given by ambiguous terms that contain a double meaning such as cock. In the Wikipedia glove you can find a more scientific denotation taking as similar words other animals while in the Twitter glove you have a more vulgar denotation taking as similar words slang and insults. Below you can find the relevant example regarding this bias difference.



```
Words most similar to bias words (Wikipedia GloVe):
Words most similar to 'cock':
1: (0.590) rooster
2: (0.516) cocks
3: (0.503) tooth
4: (0.491) cat
5: (0.486) claw
6: (0.485) rabbit
7: (0.472) knuckle
8: (0.468) tarantula
9: (0.466) rat
10: (0.466) coyote
11: (0.465) dragon
12: (0.463) sparrer
13: (0.463) boar
14: (0.461) fights
15: (0.460) bull
16: (0.459) buster
17: (0.459) canine
18: (0.452) wag
19: (0.448) radula
20: (0.448) goossen
```

Figure 9: Cock in Wiki glove



```
Words most similar to bias words (Twitter GloVe):
Words most similar to 'cock':
1: (0.801) dick
2: (0.793) tits
3: (0.767) pussy
4: (0.763) sucking
5: (0.730) cocks
6: (0.727) dildo
7: (0.717) arse
8: (0.713) milf
9: (0.709) balls
10: (0.708) anal
11: (0.708) lick
12: (0.707) slut
13: (0.701) hairy
14: (0.699) boobs
15: (0.698) cunt
16: (0.693) clit
17: (0.691) porn
18: (0.690) butt
19: (0.684) dicks
20: (0.676) licking
```

Figure 10: Cock in Twitter glove

Can biases between concepts be revealed through visualizations?

Yes, biases can be revealed through visualizations such as word embedding plots, where you can map word vectors in a two-dimensional space using dimensionality reduction techniques like PCA (Principal Component Analysis) or t-SNE (t-distributed Stochastic Neighbor Embedding). This would allow you to see clusters of words that are closely related in the embedding space and identify biased groupings or associations. An example of this visualization using PCA is provided below.

From the two plots is possible identify clusters and spatial relationships that indicate biases between concepts in the word embeddings.

- **Gender Bias**: In both embeddings is possible to see that "man" and "woman" are relatively close to each other, indicating that they are understood to be opposites or complementary concepts in these embeddings. However in the Twitter GloVe plot, "man" and "woman" are slightly farther apart than in the Wikipedia GloVe plot, which may suggest a higher degree of differentiation or stereotype in the Twitter data.
- **Ethnic Bias**: In both visualizations, "European" and "African" are positioned relatively far apart, particularly in the Twitter embedding. This distance suggests a clear separation in how these ethnicities are conceptualized, reflecting potential bias in representation.

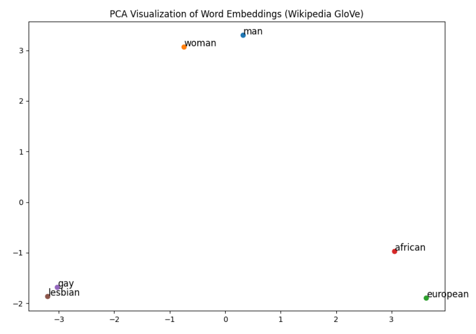


Figure 11: PCA visualization of Wikipedia Bias

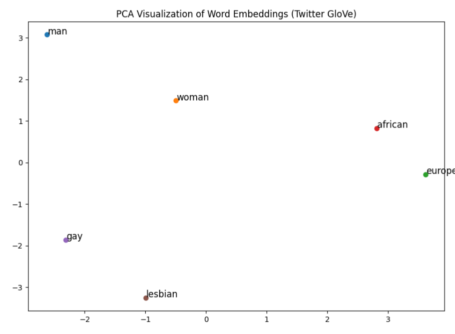


Figure 12: PCA visualization of Twitter Bias

- **Sexual Orientation Bias**: In both plots "gay" and "lesbian" are closely clustered together, which indicates that these concepts are perceived as highly similar in these embeddings. While this might seem neutral, the close clustering and context in which these terms appear in embeddings often bring with them stereotypes and biases as was indicated in the word similarity results.

In conclusion, visualization can help detect biases between concepts but as seen for sexual orientation biases, other techniques should be added to find hidden connections that reside in the word embedding itself.

Can bias in word embeddings be removed? If so, how?

The complete removal of biases in word embeddings is likely impossible, as it requires addressing deeply ingrained stereotypes that exist within society. Surely techniques can be adopted that aim to reduce these biases, but they will not entirely eliminate the underlying preconceptions that are often embedded in human cognition and societal norms. These biases reflect how individuals perceive and interpret the world, making the task of completely eradicating them from word embeddings an extremely complex if not impossible challenge.

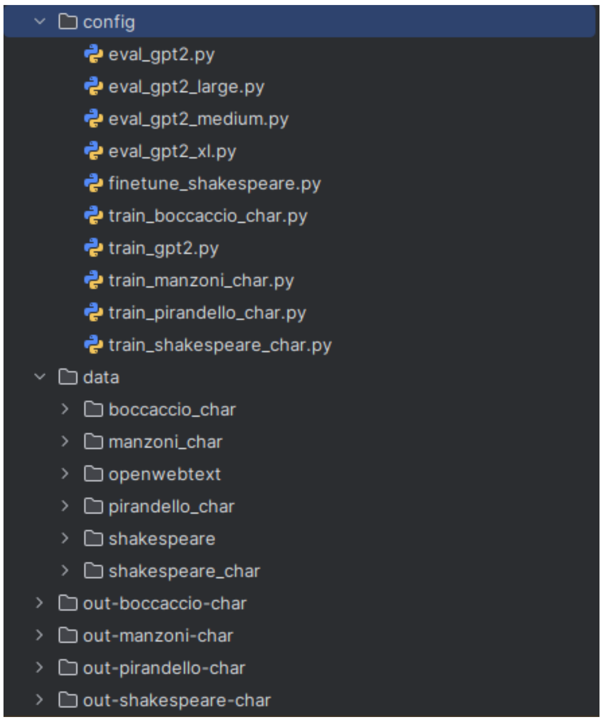
2 Your Own Personal GPT

This section provides the motivation behind training nanoGPT, details about the training process using different corpora, and the various hyper-parameters used. The code was also modified to ensure that the generated text could be saved persistently. The modified code is available here.

Motivation Behind the Chosen Corpus

The motivation for choosing these particular corpora was to observe how nanoGPT adapts to learning Italian and reproduces the language's style. To do so, we trained models on three different corpora from Italian literature, each from a distinct era. The authors—Boccaccio, Manzoni, and Pirandello—each represent a different historical period, providing a variety of writing styles, especially since the Italian language was still evolving in the Middle Ages. The datasets for Boccaccio and Manzoni were available directly from HuggingFace, which you can access here. For Pirandello, different works were combined into a single corpus using a bash script.

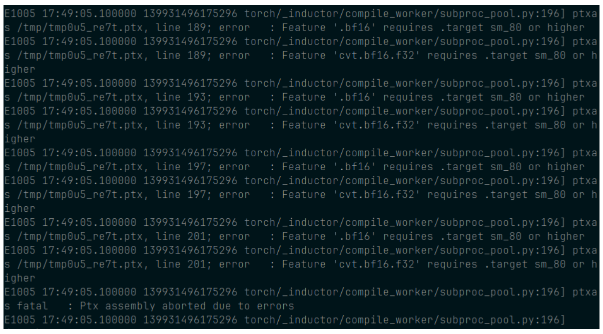
To select and make these corpora persistent, the structure of the nanoGPT repository was modified slightly, including the addition of a 'prepare.py' script for all chosen corpora in 'data/', and configuration files named 'train_author' in the 'config/' directory, as shown in Figure 13.



```
config
    eval_gpt2.py
    eval_gpt2_large.py
    eval_gpt2_medium.py
    eval_gpt2_xl.py
    finetune_shakespeare.py
    train_boccaccio_char.py
    train_gpt2.py
    train_manzoni_char.py
    train_pirandello_char.py
    train_shakespeare_char.py
data
    boccaccio_char
    manzoni_char
    openwebtext
    pirandello_char
    shakespeare
    shakespeare_char
out-boccaccio-char
out-manzoni-char
out-pirandello-char
out-shakespeare-char
```

Figure 13: Modified nanoGPT directory

The outputs produced by nanoGPT were made persistent and saved in directories



```
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 189; error   : Feature '.bf16' requires .target sm_80 or higher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 189; error   : Feature 'cvt.bf16.f32' requires .target sm_80 or h
igher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 193; error   : Feature '.bf16' requires .target sm_80 or higher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 193; error   : Feature 'cvt.bf16.f32' requires .target sm_80 or h
igher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 197; error   : Feature '.bf16' requires .target sm_80 or higher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 197; error   : Feature 'cvt.bf16.f32' requires .target sm_80 or h
igher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 201; error   : Feature '.bf16' requires .target sm_80 or higher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s /tmp/tmp0u5_re7t.ptx, line 201; error   : Feature 'cvt.bf16.f32' requires .target sm_80 or h
igher
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196] ptxa
s fatal   : Ptx assembly aborted due to errors
E1005 17:49:05.100000 139931496175296 torch/_inductor/compile_worker/subproc_pool.py:196]
```

Figure 14: Example of error encountered during gpu training

named ‘out-author-char‘. Specifically, the selected works were: - **Boccaccio**: "The Decameron," a key 14th-century work. - **Manzoni**: "The Betrothed," one of the most renowned Italian novels. - **Pirandello**: A combination of "Henry IV," "The Late Mattia Pascal," "The Rules of the Game," and "Six Characters in Search of an Author." The combined corpus was created using the following bash command:

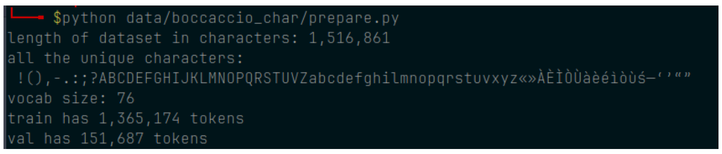
```
cat Enrico\ IV\ -\ Luigi\ Pirandello.txt \
Il\ fu\ Mattia\ Pascal\ -\ Luigi\ Pirandello.txt \
Il\ giuoco\ delle\ parti\ -\ Luigi\ Pirandello.txt \
Sei\ personaggi\ in\ cerca\ d\'autore\ - \Luigi\ Pirandello.txt \
> Pirandello.txt
```

Training and Outputs

The initial training attempt used GPU acceleration; however, due to hardware limitations—using a GTX 1650 without tensor cores—and software limitations (e.g., requiring ‘.target sm_80‘ when the hardware only supported ‘sm_75‘), I had to adjust my approach, this error is shown belowe in Figure 14. Specifically, I modified ‘train.py‘ to train using ‘float16‘ instead of ‘bfloat16‘. However, training on this GPU was still extremely time-consuming, so I ultimately switched to CPU training, despite the performance limitations.

Boccaccio Corpus

The Boccaccio corpus was selected to train nanoGPT on "The Decameron", one of the most influential works of the 14th century. The motivation behind choosing this text was to observe how nanoGPT handles the archaic style of medieval Italian and adapts to its syntax and vocabulary. As shown earlier, the Boccaccio corpus contained 1,516,861 characters, with 76 unique characters in the vocabulary, consisting of letters, punctuation marks, and accents typical of the Italian language (see Figure 15).



```
└──■ $python data/boccaccio_char/prepare.py
length of dataset in characters: 1,516,861
all the unique characters:
 !(),-.:;?ABCDEFGHIJKLMNOPQRSTUVZabcdefghilmnopqrstuvxyz«»ÀÈÌÒÙàèéìòùś–‘’“”
vocab size: 76
train has 1,365,174 tokens
val has 151,687 tokens
```

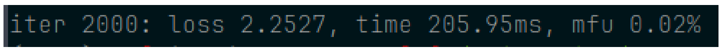
Figure 15: Boccaccio corpus statistics

The training for the Boccaccio corpus involved several iterations of experimenting with different hyperparameters. The baseline model used the following configuration:

Learning Rate: 0.001 (default) Number of Layers (n_layer): 4 Number of Attention Heads (n_head): 4 Embedding Size (n_embd): 128 Batch Size: 12 Dropout: 0.0 Block Size: 64 Max Iterations: 2000 Learning Rate Decay Iterations: 2000 The training command used for this experiment was:

```
 python train.py config/train_boccaccio_char.py --device=cpu --compile=False --eval_i
--log_interval=1 --block_size=64 --batch_size=12 --n_layer=4 --n_head=4 --n_embd=128
--max_iters=2000 --lr_decay_iters=2000 --dropout=0.0
```

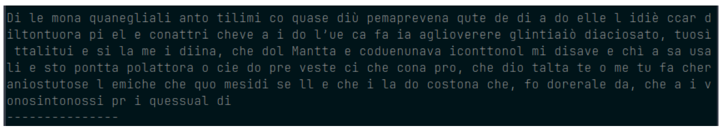
Training was conducted on a CPU, which resulted in a slower training process, but the overall goal was still achieved. After 2000 iterations, the final training loss was recorded at 2.2527, indicating that the model was making progress in learning the structure of the Italian language.



```
iter 2000: loss 2.2527, time 205.95ms, mfu 0.02%
```

Figure 16: Training progress at iteration 2000

Despite the steady decrease in training loss, the outputs generated after 2000 iterations were still somewhat nonsensical. This is typical for character-level GPT models when trained on a small dataset for a relatively short number of iterations. The model demonstrated an ability to generate valid Italian characters and some sentence-like structures, but the semantic meaning was not yet coherent. An example of the generated text after training is shown in Figure 17. While the text contains recognizable Italian words and uses punctuation appropriately, the sentences do not carry meaning and are mostly random sequences of characters and words. This output suggests that while the model has begun to understand the basic structure of Italian writing, it would require additional training iterations or more advanced techniques to produce meaningful and coherent text.



```
Di le mona quanegliali anto tilimi co quase diù pemaprevena qute de di a do elle l idiè ccar d
iltontuora pi el e conattri cheve a i do l’ue ca fa ia aglioverere glintiaiò diaciosato, tuosì
 ttalitui e si la me i diina, che dol Mantta e coduenunava iconttonol mi disave e chì a sa usa
li e sto pontta polattora o cie do pre veste ci che cona pro, che dio talta te o me tu fa cher
aniostutose l emiche che quo mesidi se ll e che i la do costona che, fo dorerale da, che a i v
onosintonossi pr i quessual di
---------------
```

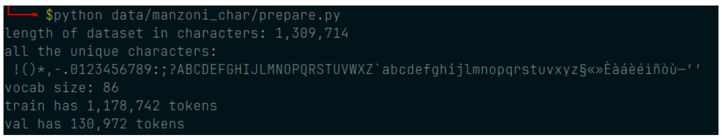
Figure 17: Generated text output from the Boccaccio corpus after 2000 iterations

Example of generated text:

```
Di le mona quanengli alianto tilimi co quase diù pemaprevena qute de di a
do elle l idiè ccar d ilton tuora pi el e conattri cheve a i do l’uve ca fa ia
aglioverere glintia iò diaciosoato, tuosi ittali tuit e si la me i diina, che
dol Mantta e coduenunava iconttonol mi disave e chi a sa usa Li e sto pontta
polattora o cie do pre veste ci che cona pro, che dio talta te o me tu fa cher
aniostutose l emiche che quo mesidi se ll e che i la do costona che, fo dorerale
da, che a i v onosintonossi pr i quessual di
```

Manzoni Corpus

The Manzoni corpus was selected to train nanoGPT on "The Betrothed" (I Promessi Sposi), a classic 19th-century Italian novel. The goal was to observe how nanoGPT handles more modern Italian syntax and vocabulary while still grappling with the challenges of literary language. The Manzoni corpus contained 1,309,714 characters and a vocabulary size of 86 unique characters, as shown in Figure 18.



```
└──▪ $python data/manzoni_char/prepare.py
length of dataset in characters: 1,309,714
all the unique characters:
 !()*,-.0123456789:;?ABCDEFGHIJLMNOPQRSTUVWXZ`abcdefghijlmnopqrstuvxyz§«»Èàáèéìñòù–‘’
vocab size: 86
train has 1,178,742 tokens
val has 130,972 tokens
```

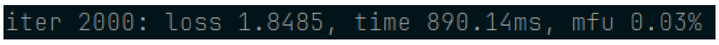
Figure 18: Manzoni corpus statistics

The training process for the Manzoni corpus followed a similar approach to the Boccaccio training, though with slight modifications in the learning process to better suit the medium-sized dataset. The model configuration was as follows:

Learning Rate: 0.001 (default) Number of Layers (n_layer): 4 Number of Attention Heads (n_head): 4 Embedding Size (n_embd): 128 Batch Size: 12 Dropout: 0.0 Block Size: 64 Max Iterations: 2000 Learning Rate Decay Iterations: 2000 The training command used for this experiment was:

```
 python train.py config/train_manzoni_char.py --device=cpu --compile=False --eval_ite
--log_interval=1 --block_size=64 --batch_size=12 --n_layer=4 --n_head=4 --n_embd=128
--max_iters=2000 --lr_decay_iters=2000 --dropout=0.0
```

Training was again performed on a CPU due to hardware limitations. After 2000 iterations, the final loss was reduced to 1.8485 (see Figure 19), indicating a significant improvement in the model’s ability to predict character sequences compared to the initial loss.

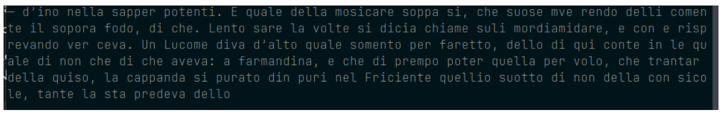


```
iter 2000: loss 1.8485, time 890.14ms, mfu 0.03%
```

Figure 19: Training progress at iteration 2000 for the Manzoni corpus

The generated output from the Manzoni corpus after 2000 iterations is shown in Figure 20. The output shows that the model is improving in generating realistic Italian text.

It can form coherent sentences and even create complex sentence structures, although the meaning is still largely nonsensical.



```
— d'ino nella sapper potenti. E quale della mosicare soppa si, che suose mve rendo delli comen
te il sopora fodo, di che. Lento sare la volte si dicia chiame suli mordiamidare, e con e risp
revando ver ceva. Un Lucome diva d'alto quale somento per faretto, dello di qui conte in le qu
ale di non che di che aveva: a farmandina, e che di prempo poter quella per volo, che trantar
della quiso, la cappanda si purato din puri nel Friciente quellio suotto di non della con sico
le, tante la sta predeva dello
```

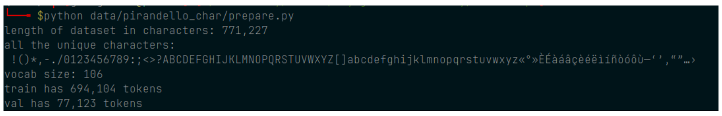
Figure 20: Generated text output from the Manzoni corpus after 2000 iterations

Example of generated text:

```
d'ino nella sapper potenti. E quale della mosicare soppa sì, che suose mve
rendo delli comen te il sopora fodo, di che. Lento sare la volte si dicia
chiamo suli mordiamidare, e con e risp revando ver ceva. Un Lucome diva
d'alto quale somento per faretto, dello di qui conte in le qu ale di non che
di che aveva: a farmandina, e che di premppo poter quella per volo, che
trantar della quiso, la cappanda si purato din puri nel Friciente quellio
suotto di non della con sico le, tante la sta predeva dello
```

Pirandello Corpus

The Pirandello corpus was created by combining several of Luigi Pirandello's works, including "Henry IV," "The Late Mattia Pascal," "The Rules of the Game," and "Six Characters in Search of an Author." This allowed for a diverse set of texts that showcase Pirandello's modernist and sometimes fragmented style. The Pirandello corpus contained 771,227 characters and a vocabulary size of 106 unique characters, including special characters and accents, as shown in Figure 21.



```
└──▪ $python data/pirandello_char/prepare.py
length of dataset in characters: 771,227
all the unique characters:
 !()*,-./0123456789:;<>?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]abcdefghijklmnopqrstuvwxyz«°»ÈÉàáâçèéëìíñòóôù—‘’,“”…›
vocab size: 106
train has 694,104 tokens
val has 77,123 tokens
```

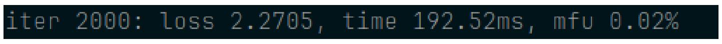
Figure 21: Pirandello corpus statistics

The training process for the Pirandello corpus used similar baseline hyperparameters to the Boccaccio and Manzoni experiments. However, given the smaller size of the Pirandello corpus, it was important to closely monitor for signs of overfitting. The model configuration was as follows:

Learning Rate: 0.001 (default) Number of Layers (n_layer): 4 Number of Attention Heads (n_head): 4 Embedding Size (n_embd): 128 Batch Size: 12 Dropout: 0.0 Block Size: 64 Max Iterations: 2000 Learning Rate Decay Iterations: 2000 The training command used for this experiment was:

```
 python train.py config/train_pirandello_char.py --device=cpu --compile=False --eval_
--log_interval=1 --block_size=64 --batch_size=12 --n_layer=4 --n_head=4 --n_embd=128
--max_iters=2000 --lr_decay_iters=2000 --dropout=0.0
```

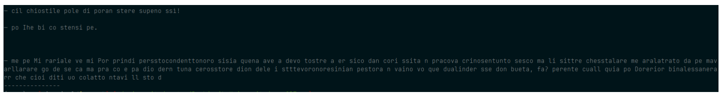
The training was again conducted on a CPU, and after 2000 iterations, the final loss was 2.2705 (Figure 22). This was slightly higher than the losses observed for Boccaccio and Manzoni, likely due to the smaller dataset and more fragmented writing style in the Pirandello corpus.



```
iter 2000: loss 2.2705, time 192.52ms, mfu 0.02%
```

Figure 22: Training progress at iteration 2000 for the Pirandello corpus

The text generated by the model after 2000 iterations, as shown in Figure 23, demonstrates that the model is able to generate coherent text in terms of sentence structure and punctuation. However, similar to the previous experiments, the text is mostly gibberish in terms of meaning, indicating that while the model can mimic the syntax and flow of Italian, it has not yet grasped the semantic aspects of the language.



```
- cil chiostile pole di poran stere supeno ssi!

- po Ihe bi co stensi pe.


- me pe Mi rariale ve mi Por prindi persstocondenttonoro sisia quena ave a devo tostre a er sico dan cori ssita n pracova crinosentunto sesco ma li sittre chesstalare me aralatrato da pe mav
arllarare go de se ca ma pra co e pa dio dern tuna cerosstore dion dele i stttevoronoresinian pestora n vaino vo que dualinder sse don bueta, fa? perente cuall quia po Dorerior binalessanera
rr che cioi diti uo colatto ntavi ll sto d
---------------
```

Figure 23: Generated text output from the Pirandello corpus after 2000 iterations

Example of generated text:

```
cil chiostile pole di poran stere supeno ssi!

po Ihe bi co stensi pe.

me pe Mi rariale ve mi Por prindi persstocondenttonoro sisia quena
ave a devo tostre a er sico dan cori ssita n pracova crinosentunto
sesco ma li sittre chessstalare me aralatrato da pe mavo...
```

Analysis and Comparison

Reflecting on the training process across different corpora with varying hyperparameter configurations provides insights into the importance of hyperparameters, the influence of the corpus size and style, and potential improvements for future exploration.

Importance of Hyperparameters in Training GPT Models

The role of hyperparameters was critical in the training of character-level GPT models, as each setting had a clear impact on the model's performance and its ability to converge effectively. The learning rate, for instance, played a significant role in controlling how the model adjusted its weights during training. Regarding model architecture, the number of layers and attention heads was kept consistent throughout the experiments, with 4 layers and 4 heads used for each corpus. This setup worked well for the larger datasets such as Boccaccio and Manzoni, as it allowed the model to capture more complex relationships in the text. The choice of batch size also influenced training stability. A batch size of 12 was selected across all experiments, which introduced an element of regularization by adding noise to the gradient updates, preventing the model from overfitting too early.

Ultimately, fine-tuning these hyperparameters based on the size and nature of the dataset was essential for achieving balanced performance. The effectiveness of each hyperparameter was closely tied to the specific characteristics of the corpus, illustrating that there is no one-size-fits-all configuration.

Influence of Corpus Size and Nature on Results

The size and nature of the corpus had a significant impact on the model's ability to generate coherent text. The largest dataset provided enough data for the model to learn complex patterns in the Italian language. This allowed the model to produce text that, while not semantically meaningful, exhibited coherent syntax and structure. The medium dataset, on the other hand, gave the model more room to generalise effectively. It is therefore inferred that with larger corpuses, the model was able to learn Italian syntax and grammar to a similar extent. However, with a smaller corpus, the model struggled to generalise effectively and the outputs it generated were more chaotic and less coherent. The fragmented and modernist writing style of Pirandello's corpus likely contributed to this difficulty, along with the smaller size, which limited the model's ability to learn coherent patterns.

In conclusion, the size of the corpus directly influenced the model's ability to generalize and generate structured text, while the literary style of each corpus affected the nature of the outputs. Larger datasets allowed the model to produce more coherent and realistic sentence structures, whereas smaller and more fragmented datasets led to less structured and more chaotic outputs.

Further Tweaks and Tests for Future Exploration

Based on the experiments conducted with Boccaccio, Manzoni, and Pirandello, several insights emerged regarding the model's behavior and the impact of different hyperparameters. First and foremost, GPU acceleration should be prioritized for future experiments. Training on a CPU, while sufficient for these initial experiments, significantly limited the speed of training and the ability to experiment with more extensive hyperparameter searches. With GPU hardware, training time would be greatly reduced, allowing for longer training sessions, increased iterations, and deeper exploration of hyperparameter configurations. Another promising avenue for exploration would involve expanding the dataset by combining a variety of Italian literary works into a single, large corpus. This unified corpus could cover different historical periods and writing styles, drawing from authors such as Dante, Petrarch, Leopardi, and others, alongside Boccaccio, Manzoni, and Pirandello. This combined corpus would serve as a more comprehensive representation of Italian literature, and training a model on this dataset would offer unique opportunities to observe how the model adapts to multiple literary styles. This would be particularly interesting because it would test whether the model can distinguish between or integrate different writing styles. For example, would the model switch between the structured, archaic style of Boccaccio and the more modern, introspective voice of Pirandello? By examining the generated outputs, it would be possible to understand how well the model handles such stylistic diversity, or if it tends to favor certain patterns over others. In summary, future exploration should focus on leveraging GPU hardware for efficient and prolonged training, while experimenting with a more extensive corpus that encapsulates the diversity of Italian literary history. This approach would provide deeper insights into how such a model adapts to stylistic nuances to gain a better understanding of the model's strengths and limitations in capturing and reproducing complex literary patterns.