Assignment 2

T-725-MALV, Málvinnsla - Natural Language Processing

Reykjavik University - School of Computer Science, Menntavegi 1, IS-101 Reykjavík, Iceland

Damiano Pasquini
damiano23@ru.is

13. October 2023

Contents

1 NLTK: The Icelandic Gold Standard

In this first part of the second assignment, it is asked to use the NLTK [3] Python library to get into practice on obtaining information over a corpus[1].

The first point asked, using the class *TaggedCorpusReader* to read the file "MIM-GOLD.sent", to display the number of sentences (done by calling **len(mim_gold.sents())**) and the individual tokens of sentence no.100 (done by calling **len(set(mim_gold.words()))**).

The second point was about displaying the total number of tokens and the number of types in the corpus: this is done by calling **len(set(mim_gold.words()))**.

The third point asked which were the ten most frequent tokens in the corpus. To accomplish this, we are asked to calculate the frequency distribution for all the words in *MIM-GOLD.sent*, returning the most common ten. This is done with the command: **most_common_10 = FreqDist(mim_gold.words()).most_common(10)**, then from the list of tokens, we take the token itself at the 0 position and the relative frequency at the position 1 to display them.

For the fourth task, aiming to visualize the twenty most frequent *Part of Speech* (PoS) tags in the corpus, is calculated the frequency distribution of **PoS tags** in the text. It uses the FreqDist function to count the occurrences of each PoS tag. The code then iterates through each word and its corresponding PoS tag in the mim_gold corpus. Then with a loop, the twenty most frequent tags and their respective frequencies are printed.

Finally, the ten most frequent tags following the tag "af" are determined by creating a list of PoS tags, then the bigrams from the PoS tags are used to calculate the *Conditional Frequency Distribution* that is useful to identify the ten most frequent occurring PoS tags that typically follow the "af". The generated output is the same as suggested in the specifications.

2 NLTK: PoS tagging

The process of classifying words into their parts of speech and labelling them accordingly is known as part-of-speech tagging, PoS-tagging, or simply tagging. Parts of speech are also known as word classes or lexical categories. The collection of tags used for a particular task is known as a tagset[1]. This second section describes various experiments over different kinds of Part of Speech taggers taken from the *NLTK library* using the *Penn Treebank corpus*.

The first two experiments asked to construct four taggers, respectively *AffixTagger, UnigramTagger, BigramTagger* and a *TrigramTagger*. Here the UnigramTagger gained the best accuracy over the test set while the worst was the TrigramTagger.
The second experiment was instead about reconstructing the same taggers, but with the "backoff" option, meaning that each tagger is assigned as *backoff tagger* to the next one. The results show that the accuracies of the backed-off taggers are higher, as we can see in Table 1.

Specifically, in the case of the BigramTagger using a UnigramTagger as a backoff tagger, the improvements are noticeable (from 13.47% to 90.8%). The reason is that the latter model can handle previously unseen word combinations and provide better tagging for rare or ambiguous words by falling back to the UnigramTagger when necessary. This combination of taggers leverages both context (bigram) and individual word statistics (unigram), resulting in higher accuracy.

[1]The complete code can be found in the GitHub repository at: `https://github.com/damiano00/T_725_MALV_Natural_Language_Processing/tree/main/assignment_2`.

Tagger	Without Backoff (%)	With Backoff (%)
Affix Tagger	27.56	
Unigram Tagger	86.08	89.85
Bigram Tagger	13.47	90.08
Trigram Tagger	8.06	90.72
Default Tagger	89.15	

Table 1: Comparison of tagger accuracies without and with backoff, and accuracy of the default NLTK tagger over the test set.

3 Your Own GPT

In this last part of the assignment, it is asked to create three small **character-level GPT models** using the *Karpathy's nanoGPT* repository, choosing a small corpus for training. It aims to experiment with the nanoGPT model to explore its capabilities. The first point is focused on choosing the right corpus, which has to have the searched characteristics (for that part it's used the *Kaggle* platform dedicated to machine learning datasets). The second point asks to train three modified GPT models using different sets of hyperparameters. Finally, the third point proposes an analysis of the work done.

3.1 Chosen Corpus

The chosen corpus to train nanoGPT[2] is referred to as the "*Supreme Court Dialogs Corpus.*"[2][3]. Although initially expansive in size, it underwent preprocessing procedures resulting in a reduction to approximately one million words. With 51,498 utterances making up 50,389 conversational exchanges, this specific corpus was chosen with the intention of ensuring that the model's outputs are contextually appropriate when it comes to sentences related to court trial dialogues. This corpus is distributed in .jsonl format which is a different version of json where each line is a single json object. Each line contains more information about the dialogues like the *case_id*, the *conversation_id*, the *text* and the *speaker*, but only the last two are used to train the neural network as we can se in Table 2.
The primary focus here is to generate meaningful and **contextually relevant exchanges** between two individuals in a court trial scenario.
The following three models are basically describing how the manual fine-tuning of the hyperparameters was done, and the improvements gained between one model and the next one. These three are not the only trained models during the fine-tuning phase, but they are the most representative of the goals achieved and how it is achieved.

Speaker	Text
"j__earl_warren":	"Number 71, Lonnie Affronti versus United States of America. Mr. Murphy."
"harry_f_murphy":	"May it please the Court. We are here by writ of certiorari to the Eighth Circuit. There is one question to be decided in this case, decided carefully. Upon sentence to consecutive ..."

Table 2: Example of dialogue between two speakers in the "Supreme Court Dialogue" corpus.

[2] `https://github.com/karpathy/nanoGPT`
[3] `https://confluence.cornell.edu/display/llresearch/Supreme+Court+Dialogs+Corpus`

Hyperparameters	Default	model_1	model_2	model_3
eval_iters	20	20	20	20
log_interval	1	10	10	10
block_size	64	**128**	**273**	**256**
batch_size	12	**12**	**80**	**32**
n_layer	4	10	10	10
n_head	4	4	4	4
n_embd	128	**512**	**512**	**512**
max_iters	2,000	7,000	10,000	5,000
lr_decay_iters	2,000	7,000	10,000	5,000
dropout	0.0	0.0	0.0	0.0

Table 3: Hyperparameters used to train the three reported models.

3.2 Model one

The first trained model was relatively the fifth training, with some modifications of the hyperparameters compared to the default configuration proposed in the GitHub repository of NanoGPT. As we can see in Table 3, the different parameters compared to the default ones are the *block_size*, the *n_embd* and the *max_iters* (consequently also in change of *lr_decay_iters*). The **block size** is increased to 128 only as a test because in the initial training, the results were most random words, and it is combined with the increase of the **number of layers** to 10 layers since it could be a good balance between computational complexity, time of training and results. Then the third modified parameter is the **number of embeddings** set to 512, which is a high value for this parameter. This was chosen to improve the details since the previous outputs were mostly random and meaningless words. Finally, for this training, the number of **maximum iterations** and **decay iterations** were set to 7000 trying to improve results. The generated dialogue was of this form:

Generated output

oscar_h_davis: has a little interence show.. The court to vacable computation.. I sure in or the way famuals employees, had jurisdicted may both have different just a charge case of deting to but Mr. Chief Metald_moseley entitledge: The question of hot to change offers –

richard_b_macguineas: Well, Your Honor.

j__felix_frankfurter: – in your proceeding what he – part of perhaps for the opinion the actual – union of are difference toward, afters it moves.. At you persuade at the Government of

As we can notice, a lot of English words are random and meaningless, but at least there is the structure **speaker-text** the GPT model is trained for. About the training phase, using a GPU with CUDA integration it took 11 minutes to train the model, so a relatively short time. During the training phase, any particular kind of difficulty emerged.

Listing 1: Script executed to train the first reported GPT model

```
python train.py config/train_shakespeare_char.py --device=cuda --
   compile=False --eval_iters=20 --log_interval=10 --block_size
   =128 --batch_size=12 --n_layer=10 --n_head=4 --n_embd=512 --
   max_iters=7000 --lr_decay_iters=7000 --dropout=0.0
```

3.3 Model two

After having faced the results from the previous model, as meaningless and random words, three hyperparameters were modified: the **block_size**, the **batch_size** and the two parameters related to the number of **iterations** (*max_iters* and *lr_decay_iters*). The block size is one of the most influential parameters, and it was chosen after having calculated the average size of each line within the dialogues since we want the expected output to be as similar as possible to the training corpus dialogues. The average characters per line were 273, so *block_size* for that second model is set to 273. Since increasing the *batch_size* makes the model more specific on the corpus it is trained, and less generalized on other data, increasing it to 80 could reduce the randomness and make more context-relevant phrases.
The last modification from the previous training is increasing the maximum number of iterations up to $10,000$. This led to having to interrupt the training to the last checkpoint after $1,000$ iterations and more than three hours probably due to the too high *batch_size* (80 instead of 12). As a consequence, it resulted in a sample quite similar to the previous one, with a lack of context, meaningless words and wrong punctuation.

Generated output

j__stanley_reed: – the – whether there was the Coast Guaranty in blanks of Eduff some term

max_fink: Well, oh, I have a mines you appeal.. And arising in its opposition that – it was in criminal impact consideration citentily permit.

j__stanley_reed: (Voice Overlap) –

max_fink: – the – the clot of any purposes so

Listing 2: Script executed to train the second reported GPT model

```
python train.py config/train_shakespeare_char.py -- device=cuda --
   compile=False --eval_iters=20 --log_interval=10 --block_size
   =273 --batch_size=80 --n_layer=10 --n_head=4 --n_embd=512 --
   max_iters=10000 --lr_decay_iters=10000 --dropout=0.0
```

3.4 Model three

After having analyzed how the previous hyperparameters performed, four of them were modified: the **block_size**, the **n_embd** and the **iterations**. The block size is decreased to the nearest power of two to 273, so 256. The number of embeddings is decreased to the previous power of two to 512, so 256, as a compromise between training time and an acceptable result. The maximum number of iterations is then decreased to $5,000$, which turned out to be enough to obtain a good result. The resulting samples from this model gained surprising results: the grammatical errors are almost absent, and the sentences are not meaningless, but still unrelated between one sentence and the next one.

Generated output

j__felix_frankfurter: That is that – that isn't all nonsupported by improper duty as against the

david_p_findling: I don't think it's – it isn't – it isn't denied, Your Honor, but three days at the fourth of the instructions, there has been all is inevitable provisions by the Commission in order the – the requirements –

j__stanley_reed: The score grounds in your argument – under the Box Chapter X?

bessie_margolin: Yes.. We think that's all simply to remove this under that – but –

Listing 3: Script executed to train the third reported GPT model

```
python train.py config/train_shakespeare_char.py --device=cuda --
   compile=False --eval_iters=20 --log_interval=10 --block_size
   =256 --batch_size=32 --n_layer=10 --n_head=4 --n_embd=256 --
   max_iters=5000 --lr_decay_iters=5000 --dropout=0.0 --
   always_save_checkpoint=False
```

3.5 Analysis and comparison

Modifying the hyperparameters to train a GPT model is important since they can determine whether the model converges effectively, generates coherent text, and produces meaningful results. Between the modified parameters during the manual fine-tuning, the most significant seems to be the following: the **block size** because the bigger it is the more complex the model is, the **batch size** since it determines how many examples are processed together during each training iteration as we saw in the second model where it took too much time to train, and the **number of embeddings** which allows learning more detailed representations of the input.
Instead, the number of words that form the corpus has influenced the results enough. This is inferred from how often some words are repeated because a million words considered for the training were part of too few dialogues, not varying the context and consequently not differentiating words. Another aspect of the utilized small corpus is that its dimension influenced the inability to create some context between question and answer.
Further improvements this experiment needs could be training a complex model (for example with a higher block size) for an elevated number of epochs (like 10,000). One easier solution could be increasing the number of words and using more columns from the corpus, like the process ID that identifies one court dialogue from another, helping the model to improve the context between sentences.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[2] LLee-Research-Collaborators. Supreme court dialogs corpus.

[3] Edward Loper and Steven Bird. Nltk: The natural language toolkit, 2002.