HW 1: CalcGPT - Niyathi Allu, nallu002@ucr.edu

Method for encoding strings:

By default **EleutherAI/gpt-neo-1.3B** uses byte pair encoding(BPE). This is basically a subword tokenization algorithm that breaks the rare or some unknown words into subwords that are known already. After tokenization, the input is embedded into a high dimensional vector space.

Attached below are the encoding for the input prompts.

- **Baseline:**
  As mentioned in the homework, the baseline to the model is just a simple summation of two numbers **"1+2 ="** and it is tokenized into ['1','+','2','=']. Attached below is the screenshot of the given input and how it is tokenized by the tokenizer.

```
baseline
Input tokenization example:
        Prompt 0: 0+0=
        Tokenized prompt 0: ['0', '+', '0', '=']
        Prompt 1: 0+1=
        Tokenized prompt 1: ['0', '+', '1', '=']
        Prompt 2: 0+2=
        Tokenized prompt 2: ['0', '+', '2', '=']
```

- **FewShot Learning :**
  This is basically, training the model with few labeled examples. Since, we were asked to generate the outputs only for the sum of the numbers ranging from (0,50) which are around 2500 all the combinations of numbers included, the model isn't performing great. So decided to go for the few shot learning by giving the model two examples, such as **1+2 = 3 and 3+2 = 5.** Attached below is the input to the model, and how it is tokenized.

```
Input tokenization example:
        Prompt 0: 1+2=3,3+2=5,0+0=
        Tokenized prompt 0: ['1', '+', '2', '=', '3', ',', '3', '+', '2', '=', '5', ',', '0', '+', '0', '=']
        Prompt 1: 1+2=3,3+2=5,0+1=
        Tokenized prompt 1: ['1', '+', '2', '=', '3', ',', '3', '+', '2', '=', '5', ',', '0', '+', '1', '=']
        Prompt 2: 1+2=3,3+2=5,0+2=
        Tokenized prompt 2: ['1', '+', '2', '=', '3', ',', '3', '+', '2', '=', '5', ',', '0', '+', '2', '=']
```

Method for generating text :

The language model used is "**EleutherAI/gpt-neo-1.3B"** that has over 1.3Billion parameters. This is trained as a masked autoregressive model with cross entropy loss that takes input as a string and predicts the next token.

The parameters that I have used are do_sample, temperature, max_new_tokens and the min_new tokens as attached below.

```
output_batch = model.generate(
    input_ids=input_batch,
    do_sample=True,
    attention_mask=attention_mask_batch,
    temperature=0.001,
    min_new_tokens = 10,
    max_new_tokens=30
)
```

We know that with less temperatures, the highest values get accentuated and the lower values get diminished. And I have carried out the experiments with temperatures of 0.1, 0.01, 0.001.

And max_new_tokens, which are the tokens in the output sequence without the tokens in the prompt are set to 30. Min_new_tokens are set to 10.

Do_sample = True ensures that we have the next new token selected based on the probabilities assigned to each word.

Attention_mask, tells the models which tokens the model has to attend to, instead of all the tokens in the input sequences including the padded tokens.

Here the batch_size is set to 25, to speed up the generating text process.

Method for decoding strings :

Attached below is the output for just the baseline input, which is 1+0 = , as mentioned we can see some incoherent output that is being generated with the baseline equations. These output strings are decoded to get an integer.

```
Output decoding example:
        Generated text 0: 0+0=0+0=0+0=0+0=0+0=0+0=0+0=0+0=0+
        Decoded output 0: 0
        Generated text 1: 0$ and $0+1=1$.

The following lemma is a direct consequence of the definition of the $n$-th
        Decoded output 1: 0
        Generated text 2: 0+2=0+2=0+2=0+2=0+2=0+2=0+2=0+2=0+
        Decoded output 2: 0

Count :  24
Accuracy :  0.96
```

The output for the few shot learning based prompts after clipping the input from the output strings look like this 0,1+1=1,1+2=2, … as shown in the example below.

```
Output decoding example:
        Generated text 0: 0,1+1=1,1+2=2,2+2=3,3+3=5,4+4=
        Decoded output 0: 0
        Generated text 1: 1,1+1=2,1+2=3,1+3=5,1+4=7,1+5=
        Decoded output 1: 1
        Generated text 2: 3,0+3=5,0+4=3,0+5=5,0+6=3,0+7=
        Decoded output 2: 3
```

Results :

- **Baseline:**
  With the baseline, as in with the input as simple as “1+2=” the language model did not perform well. Out of all the 2500 inputs with the input numbers ranging from 0 to 50, the model is able to give correct answers only for 25 input equations.

  It is observed that with the given input and the parameters as mentioned above in the section b, the model performed very poorly and it started to give the text as the outputs as well.

  Overall the model has an **accuracy over 1%** for the baseline.

- **FewShot Learning :**
  With few shot learning, as in giving the model two labeled inputs as mentioned above included in the prompt, the model performed well as compared to the baseline.

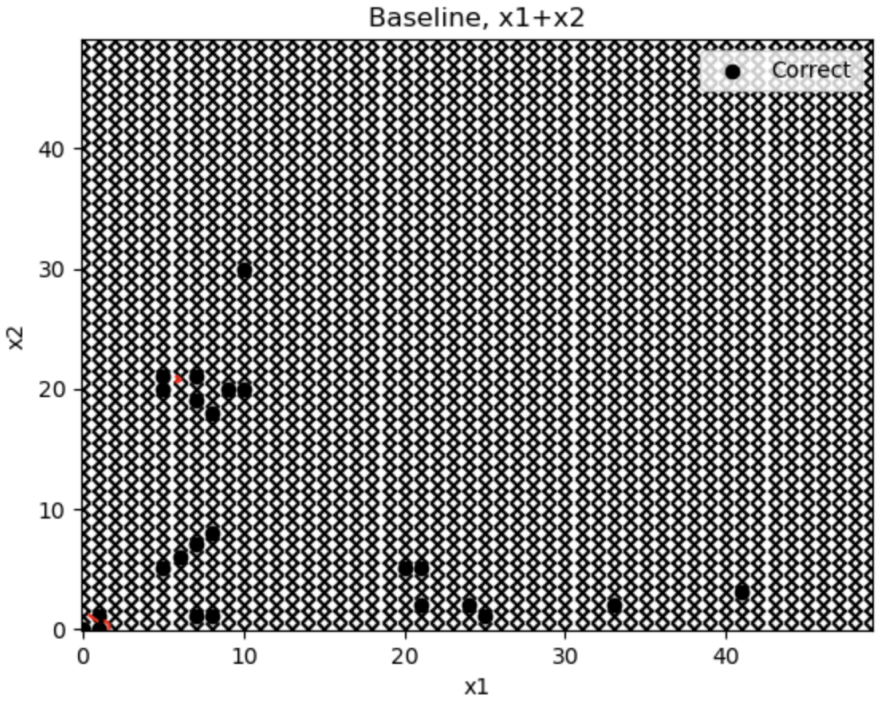
  With the few shot learning and the parameter values as mentioned above, the model got an **accuracy over 9.88%** as opposed to the 1% accuracy from the baseline model.

- **K-Mean Clustering Algorithm:**
  For the contour, I have used the K_means clustering algorithm. This K-means classifier predicted the correct outputs with an accuracy of 82% on the few shot learning prompt.

  This randomly initializes k centroids, assign each datapoint closest to the centroid with Calculated euclidean distance between the data points and the centroids. After trying out Other classifiers, it is figured that K-Means clustering best fits the scatter plot of correct And incorrectly predicted values.

```
Baseline
Count :  24
Accuracy :  0.96
```

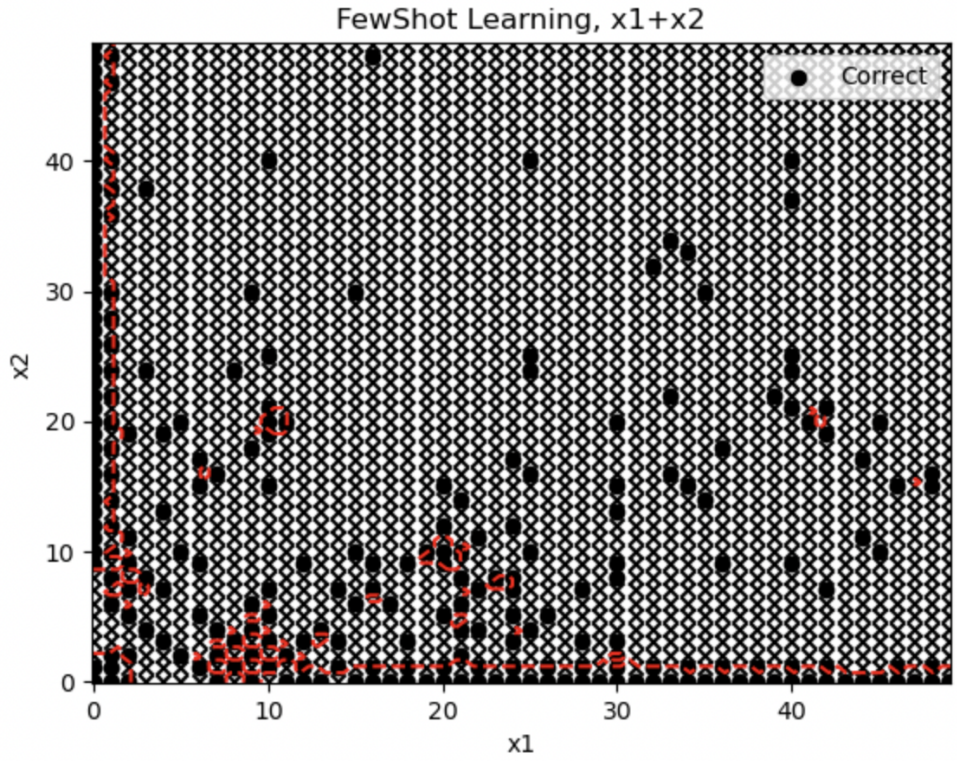
```
/Users/niyathiallu/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.py
tosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11
me False, the `axis` over which the statistic is taken will be eliminated, and the value None wil
arning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



Temperature : 0.001, Accuracy = 0.99%

```
FewShot Learning
Count :  248
Accuracy :  9.92
```

```
/Users/niyathiallu/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classifica
tosis`), the default behavior of `mode` typically preserves the axis it acts along. In Sci
me False, the `axis` over which the statistic is taken will be eliminated, and the value N
arning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



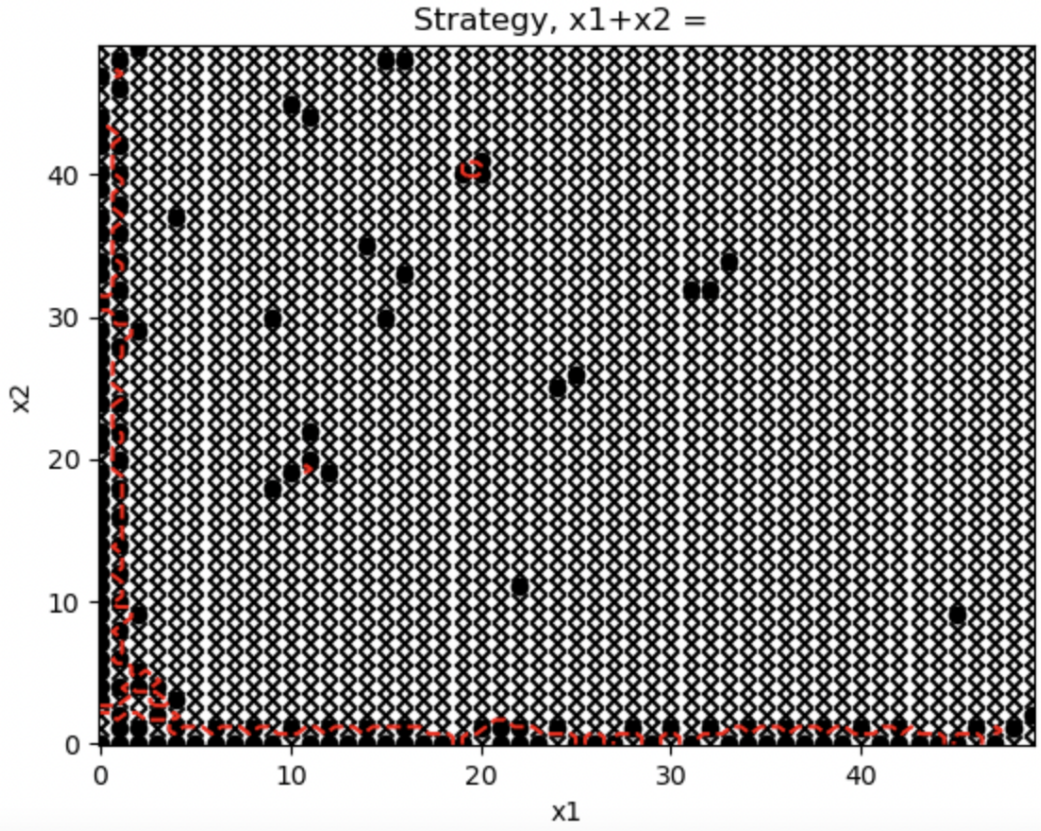
Temperature: 0.001, Accuracy = 9.92%

Extra Experiments :

- **With temperature 0.1, 0.01, 0.001 :**

  With temperature 0.1, the accuracy for the baseline has not changed much, but for a few shot learning with temperature 0.1, the accuracy came around to be 6.3%. And Attached below is the scatter plot for the sum of numbers ranging from 0 to 50 with Temperature = 0.1.

```
Count :  154
Accuracy :  6.16
/Users/niyathiallu/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_cla
tosis`), the default behavior of `mode` typically preserves the axis it acts along.
me False, the `axis` over which the statistic is taken will be eliminated, and the
arning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



And the scatter plot for the sum of numbers from 0 to 50 with the temperature 0.01 gave An accuracy of **9.63 %**. And for the temperature 0.001 the accuracy is **9.88%.** There is not a significant improvement in the accuracies here from 0.01 to 0.001. The screenshot is attached in the results section above.

- **Giving more labeled examples in the input prompt:**
  - **1+2=3,3+2=5,x1+x2 =**

    Here in this case, I have given two labeled examples only in the prompt. But few shot learning is supposed to give more accuracy, with more labeled examples in the prompt.

    With these two examples, and the temperature = 0.001 and the I got the **accuracy of 9.92%** as mentioned above.

  - **1+2=3,3+2=5,51+2,x1+x2 =**

    When I tried to include more labeled inputs, and one input in the prompt that is out of range of (0,50), I expected the accuracy to go up. But the accuracy did not go up, in fact the accuracy came out to be around **7.29%** only.

- **Other arithmetic operations:**

```
output_str, probs, output = infer("1+2,3+4=5,61*62=")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'1+2,3+4=5,61*62=5,61*63=5,61*64=5,61*65=5,61*66=5,61*67=5,61*68='
```
```
output_str, probs, output = infer("61*62=")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'61*62=0.0, *p*=0.0, *r*=0.0, *n*=10) and the control group (mean±SD: *t*=0.0, *p*'
```

```
output_str, probs, output = infer("1+2,3+4=5,10/5=")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'1+2,3+4=5,10/5=1,11/5=1,12/5=1,13/5=1,14/5=1,15/5=1,16/5='
```
```
output_str, probs, output = infer("10/5=")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'10/5=0.5\n\nA:\n\nYou can use the following code:\nimport numpy as np\n\ndef f(x):\n    return x**2\n\nx = np.array([1'
```

```
output_str, probs, output = infer("1+2,3+4=5,((10/5)+(2*3)=")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'1+2,3+4=5,((10/5)+(2*3)=6)\n\nA:\n\nYou can use the following code:\n#include <stdio.h>\n\nint main(void'
```
```
output_str, probs, output = infer("((10/5)+(2*3)")
```
```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```
```
output_str
```
```
'((10/5)+(2*3)+(2*4))\n\n(10/5)+(2*3)+(2*4)\n\n(10/5)+(2*3)+(2*4)\n'
```

- **With FineTuning of gpt2 model :**
  - https://colab.research.google.com/drive/1KlflFHGj7crizwDMiEqUcj4cLPw1XwW9?usp=sharing
  - Instead of **EleutherAI/gpt-neo-1.3B, gpt2(50261,768)** fine tuned owing to Resource crunch the collab has.
  - The dataset is split into train, test and validation.
  - The training dataset has 1600 inputs, validation has 400 and the test dataset has 500 data points.
  - The input string to the language model is of the form, <sos>x1+x2 =<ans>:<eos>

```
['<startofstaring>31+22= <answer>: 53<endofstring>',
```

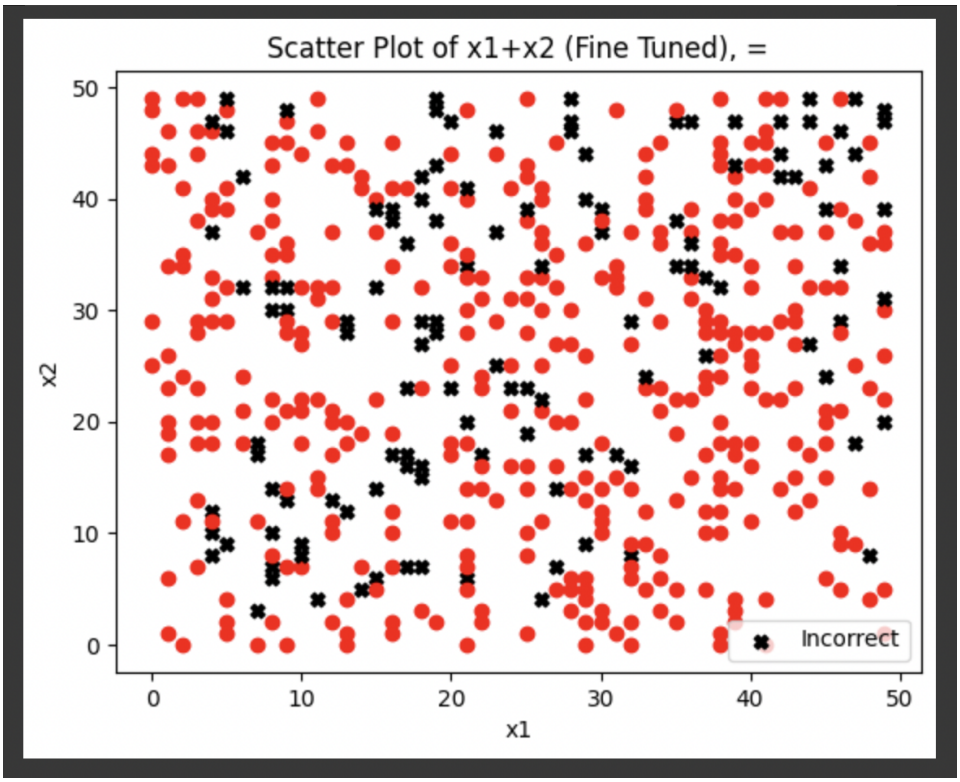
  - And the special tokens, <startofstring>,<endofstring> and tokens <answer> have been added to the tokenizer.
  - Learning rate of 5e-5, epochs = 5, and AdamW optimizer has been used to finetune the model.

```
Epoch 1 of 5
100%|          | 5/5 [01:18<00:00, 15.71s/it]
Avg training loss: 0.6834852678701282
100%|          | 5/5 [00:16<00:00,  3.28s/it]
Avg validation loss: 1.7674348503351212
Epoch 2 of 5
100%|          | 5/5 [01:13<00:00, 14.67s/it]
Avg training loss: 0.44927730910480024
100%|          | 5/5 [00:15<00:00,  3.17s/it]
Avg validation loss: 1.5703318439424039
Epoch 3 of 5
100%|          | 5/5 [01:11<00:00, 14.37s/it]
Avg training loss: 0.4042024763301015
100%|          | 5/5 [00:16<00:00,  3.25s/it]
Avg validation loss: 1.4387809790670871
Epoch 4 of 5
100%|          | 5/5 [01:11<00:00, 14.35s/it]
Avg training loss: 0.37356250774115324
100%|          | 5/5 [00:15<00:00,  3.18s/it]
Avg validation loss: 1.3571206398308278
Epoch 5 of 5
100%|          | 5/5 [01:11<00:00, 14.32s/it]
Avg training loss: 0.35406082320958376
100%|          | 5/5 [00:15<00:00,  3.19s/it]Avg validation loss: 1.3202784761786461
```

  - For the epochs = 3, the accuracy is not higher and the validation loss is high. So the number of epochs has been increased to 5.
  - In each epoch, we are training on the train dataset using a batch size of 10, calculate the loss, backpropagate, compute the gradients. After the training it is evaluated on a validation dataset using a similar process without parameter updates.
  - After training the output for the given input of **“22+25 =”** looks as attached below

```
<startofstring> 22+25 = <answer>: 52 <endofstring> 63 <endofstring> 74 <endofstring> 77 <endofstring>
<answer>: <answer>: 85 <endofstring> <answer>: <answer>: 92 <endofstring> 94 <endofstring> <startofstring>
<startofstring> <startofstring> <endofstring> <endofstring> <endofstring> <endofstring> <endofstring>
<endofstring> <endofstring> <endofstring> <endofstring> <endofstring> <endofstring> <endofstring>
<endofstring> <endofstring> <endofstring> <endofstring> <endofstring> <endofstring> <endofstring>
<endofstring> <endofstring> <endofstring> <endofstring>
```

- And the trained model has been evaluated against the test dataset and it achieved an accuracy over **75.6%. (On the test dataset, 500 data points)**
- Attached below is the scatter plot of the test dataset and the predicted values from the trained model.



- **Observations :**
  - Without fine tuning the results are incoherent with “gpt2”.

```
'22+25 = \xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-\xa0-
\xa0-\xa0-\xa0-\xa0-\xa0-\xa0'
```

```
⌘/Ctrl+M B
'What is the sum of 22 and 25? \xa0The sum of 22 and 25 is the sum of the two numbers. \xa0The
sum of 22 and 25 is the sum of the two numbers. \xa0The sum of 22 and 25 is'
```

  - This fine tuned **“gpt2**”model, as opposed to the “**EleutherAI/gpt-neo-1.3B”** without fine tuning showed a significant improvement in the accuracy for test dataset.
  - As observed the scatter plot is in quite contrast with that of few shot learning prompts.

- But one of the shortcomings is that this model, is performing similar to "**EleutherAI/gpt-neo-1.3B"** for the dataset outside the range (0,50) i.e the dataset that it is trained on.
- I believe this can be attributed to either overfitting, or the fact that the dataset used to finetune gpt2 is quite small considering how large the gpt2 model is and we need more dataset other than 1600 labeled data points for the model to perform better on data points outside its range.
- From this experiment I am confident that given we have enough resources, to fine tune "**EleutherAI/gpt-neo-1.3B,",** 1.3B parameters model can give better results when compared to the results we got with few shot learning prompts.
- And in the few shot learning for the "**EleutherAI/gpt-neo-1.3B," why are all the correct answers concentrated close to when either x1, or x2 = 0?**
- After a few shot learning also it is observed that it fails to predict answers correctly on the out of range (0,50).
- But it is observed that in **"x1+0 = "** the model gives correct answers even when x1 is out of range. It can be any answer and the model predicts correctly. I have a hard time understanding what is remembered and what the model has actually learnt. And how to measure it.

AI collaborators and references:

- **https://chat.openai.com/ (**For some doubts, documentation and to cross check the code to dataset generation,plot and debugging)
- **https://www.youtube.com/watch?v=elUCn_TFdQc&t=1458s**
- **https://towardsdatascience.com/how-to-fine-tune-gpt-2-for-text-generation-ae2ea53bc272**