

Homework 4 - CS 5787 Deep Learning - Spring 2023

Your homework submission must cite any references used (including articles, books, code, websites, services (e.g., ChatGPT, GitHub copilot, etc), and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. If you do work with others, you must list the people you worked with. Submit your solutions as a PDF to Canvas. Your homework solution must be typed, you may use LATEX if you wish. Homework must be output to PDF format. I suggest using <http://overleaf.com> (<http://overleaf.com>) if you use LATEX to create your document. Overleaf is free and can be accessed online. Note that this assignment adds up to 15% of the total grade: 150 points. In addition, there are 50 points of extra credit (e.g., if you get 200 points that means you'd get the 15% for the class, plus the equivalent of another 5% that can be applied to other HWs or to your total grade). If told to implement an algorithm, do not post your code to a public web repository (e.g., GitHub).

Part 1 - Concepts (150 points total, 10 points each)

Answers should be short (1-2 paragraphs; single sentence answers are not valid), concise, and contain sufficient technical details. No coding for this section.

1. What is a self-attention mechanism and how is it used in Transformers?

Self-attention is one of the building blocks that can be found in a transformer. Including a self-attention block in the architecture allows the model to better grasp the relationship between different elements in a sequence.

As an example, using a self-attention transformer on text data allows the model to understand the context and long-range dependencies between words much better than in RNNs or CNNs. Because of these characteristics, this mechanism achieves great results for ML tasks such as machine translation and text-summarization.

From a technical standpoint, the attention mechanism performs nothing more than a calculation on the output of each layer of the encoder and decoder. The attention block generates a matrix of scores, computed for each pair of elements in the input sequence. Such scores effectively convey the importance of certain relationship between tokens, even if far apart in the sequence, and are used to compute a weighted sum of the input embeddings, that basically score how important each input element should be in generating the output.

2. What is the difference between multi-head attention and single-head attention in Transformers?

As the name suggests, the difference between single-head and multi-head attention is in the number of attention "heads" or blocks that are included in the architecture of the Transformer.

Using multiple attention blocks in parallel, instead of a single one allows the model to assign to each attention block a different pattern or aspect to capture in the input sequence. Each attention mechanism becomes an expert in understanding the relationship between elements with respect to a specific context or experts at spotting a specific pattern. All outputs are finally linearly concatenated to generate the final output.

Ultimately, while single-head attention are great at understanding general contextual information and patterns, they lack the depth and specificity of multi-head attention models, which are much more comprehensive and detailed.

3. Suppose that we design a deep architecture to represent a sequence by stacking self-attention layers with positional encoding. What could be issues?

Stacking self-attention layers with positional encoding in a deep architecture can run in various issues, in terms of computational complexity as well as performance. Firstly, stacking layers is **quite computational expensive** and might lead to unfeasible training times. Moreover, such architectures have a **greater risk of overfitting** and are **prone to experience vanishing gradient**, which can be mitigated using methods like layer normalization and residual connections.

If dealing with long sequences, an architecture featuring stacked self-attention layers with positional encoding can **may not be able to capture complex long-range dependencies** due to the inherited limitations of positional encodings, hence require more complex techniques. Finally, it could also be **very challenging to scale**, since self-attention layers have quadratic complexity with respect to the sequence length.

4. How would you design a learnable positional encoding method?

Learnable positional encoding is a method that allows the model to better capture fine-tuned patterns and relationships across text. In designing one, I would firstly initialize the trainable matrix of positional encoding, which is of shape $(MSL \times ED)$ where MSL is the maximum length of the input sequence and ED is the dimension of an embedding. This matrix is then added to input token embeddings. This element-wise sum is dynamically updated during training as the model learns.

5. Is it a good idea to replace scaled dot-product attention with additive attention in the Transformer? Why?

It is not a good idea to replace scaled dot-product attention with additive attention in Transformers.

While additive attention is marginally more robust to the scale of input values, it does not clearly and consistently outperform dot-product attention in any observable application. Therefore, a fundamental aspect in the choice should be the computational efficiency. Dot-products are fully optimized and extremely efficient, especially when using GPUs, which is often the case for deep architectures.

Therefore, scaled dot-product attention is significantly faster than additive attention, hence should be the preferred choice in most scenarios.

6. For language modeling, should we use the Transformer encoder, decoder, or both? How would you design this method?

Language modeling is an autoregressive task, hence **the Transformer decoder should be used**. The decoder is more appropriate because is specifically designed to handle tasks where only part of the input is available, while the encoder is designed to work on tasks that need the entirety of the input sequence, such as translation.

The architecture I would design for such a model would feature a stack of left-to-right Transformer decoder layers with self-attention and normalization. The input needs to be processed in a certain way before being fed in the model. Firstly, it has to be tokenized, and populated with start-of-sequence and end-of-sequence tokens where needed. Then, positional encoding can be applied on their embeddings. Furthermore, part of the input has to be progressively masked to ensure that only past elements are used to predict the current token.

Finally, The output should feature a linear layer followed by a softmax activation function, in order to output the probability score for each token in the vocabulary to be the current token, ultimately choosing the most plausible as the final prediction.

7. What can be challenges to Transformers if input sequences are very long? Why?

The most important challenges that Transformers face when dealing with large input sequences are two.

First, the model can quickly become computationally unfeasible, since the self-attention mechanism has a quadratic complexity with respect to the input sequence length. Hence, **long sequences require high memory usage and computational power**.

Secondly, deep architectures when dealing with long sequences are particularly prone to **vanishing/exploding** gradients during training. This issue effectively means that the model fails to learn long-range dependencies because the gradients tend to zero or spike to huge values. These issues become more relevant as the sequence length increases because, with backpropagation, errors propagate and worsen through time.

8. Say that you are asked to fine tune a language model to perform text classification by adding additional layers. Where will you add them? Why?

Fine-tuning a LM by adding additional layers to perform text classification is effectively applying transfer learning to a pre-trained language model, to adapt it to your specific text-classification task.

Therefore, **layers should be added on top of the output of the language model**, so that they can keep the original architecture intact and leverage all the knowledge that the original model has acquired in terms of ability to understand semantic features and patterns in the input text.

You should build upon this foundation by adding layers specific to your classification task, that will reshape the original output format into your desired output format.

9. All other things being equal, will a masked language model require more or fewer pretraining steps to converge than a left-to-right language model? Why?

A masked language model requires less pretraining steps to converge compared to a left-to-right language model, when all other things are equivalent.

That is because, while left-to-right models are only fed tokens that precede the target token in the sequence, masked language models can also leverage the context conveyed by future tokens that are not masked. The increase in information generally allows them to learn more accurate relationships within the input text and converge faster.

10. Machine translation has long been evaluated based on superficial matching between an output translation and a ground-truth translation. How would you design a measure for evaluating machine translation results by using natural language inference?

Language translation is a complex and open field, where it does not exist a right answer. Sentences can be translated accurately in a myriad of ways, the fundamental feature of a good translation is retaining the logical implications and relationships among concepts present in the original text. Therefore, assessing the validity of a translation by matching the output translation and a ground-truth translation word by word is a very simplistic approach.

A more thorough metric is using NLI, Natural Language Inference. NLI tasks can extract features from text such as meaning, context and the relationships and logical implications mentioned above. To measure machine translation with NLI I would **compare the NLI model's performance on the output translation with the one obtained on the ground-truth**. This comparison can be then evaluated with the canonical evaluation metrics, i.e. accuracy, F1-score and so on, depending on the task.

This allows good translations that use different wording or syntactic constructs compared to the ground truth text, to still score highly.

11. How can we leverage BERT in training language models?

BERT is a large pre-trained NLP model that has been trained on a huge corpus of text, thus learning high-quality embeddings and general patterns in text. It is particularly successful at extrapolating context-based relationships in text, making it very useful for a variety of tasks.

BERT can be used to train language models by leveraging its pre-trained features. Its weights are extremely valuable as they convey that general contextual knowledge. These **weights can therefore be used to initialize an LLM and expedite training, improving performance**. The further training on the base weights borrowed from BERT will fine-tune the model to a specific task or dataset.

12. Suppose that we have a trained model based on multi-head attention and we want to prune the least important attention heads to increase the prediction speed. How can we design experiments to measure the importance of an attention head?

There are two main experiments that can be ran to estimate the importance of attention heads, with the intent to prune the least important to improve the computational performance of the model.

A very common approach is **head ablation**, that essentially consists in removing each attention head one at a time and evaluate the performance of each pruned model. The model with the best performance on the validation set is considered to be the one in which the least important attention head was pruned.

Another approach is **gradient based importance**. It consists in computing the gradients of the loss with respect to the weights of each attention head.

13. Consider sequence to sequence problems (e.g., machine translation) where the input sequence is always available throughout the target sequence prediction. What could be limitations of modeling with decoder-only Transformers? Why?

Decoders use a technique called **sequential decoding** to generate the output, which **consists in generating tokens in series**, from left to right, as if they were reading progressively through the input text. Machine translation however, is a task that allows language models to leverage the full input, without any positional masking. Therefore, sequential encoding are a sub-optimal approach as they **lack parallelization**, hence can be quite slow for long input sequences.

Using encoder-decoder architectures is recommended in sequence-to-sequence problems such as translation.

14. What is the difference between a pre-trained Transformer model and a fine-tuned Transformer model?

Pre-trained Transformer models are models that have been already trained in an unsupervised setting on often huge datasets, i.e. BERT, hence have converged to a set of weights that conveys a **general knowledge of semantic patterns** present in the training input sequences. However, they are not ready-made to be applied to every specific task that you might want the model to perform.

Fine-tuned Transformer models instead, are the result of adapting these general pre-trained models to perform such specific tasks. They are obtained by training the pre-trained models on an additional dataset, which is task-specific, smaller in size and labeled dataset. The goal of fine-tuning is leveraging the general patterns that the pre-trained model already learnt to serve as a good starting point in **mastering a more specific task**, such as sentiment analysis.

15. You are going to use a LLM for an NLP task. What are the different ways you can use it? if you fine-tune it, what are the options?

LLMs can be very useful also for NLP tasks. In particular, LLMs are very successful in extracting features from the input text, which can be general patterns or more specific relationships, depending on if the LLM has been fine-tuned for that specific task or not.

Without fine-tuning, LLMs can perform **zero-shot learning**, a NLP approach to perform tasks without any additional training, thanks to a pre-processing prompt-engineering step to phrase the question in a general fashion. **With fine-tuning**, a great option is **few-shot training**, that improves on zero-shot by leveraging the capabilities of fine-tuned LLMs to generalize a small set of specific examples to improve its understanding of the context.

Part 2 - Hands-on with LLMs (50 points)

For this problem, you may work in groups of two, and you will use an open source LLMs to run some experiments (think of this as a mini-hackathon). There are many options, but for this assignment you may use the following repository:

<https://github.com/ggerganov/llama.cpp> (<https://github.com/ggerganov/llama.cpp>)

This implementation allows you to run LLMs locally on your machine (it supports multiple models). Depending on your hardware, it can be slow, so if you prefer to use something else that uses an API you are free to do so (note that if you use an API to call an LLM you may have to pay- you should not spend more than a few dollars, if you decide to go for that option).

You may also use LangChain:

<https://python.langchain.com/en/latest/index.html> (<https://python.langchain.com/en/latest/index.html>) See <https://python.langchain.com/en/latest/modules/models/llms/integrations/llamacpp.html> (<https://python.langchain.com/en/latest/modules/models/llms/integrations/llamacpp.html>).

IF you work with a partner, your submissions for this section should be identical. You must clearly indicate at the start of the section who your partners is. You should also both set up and run your experiments (the idea is for both of you to do the work!).

This min-hackathon should consist of the following steps:

1. Get the model running for a basic task (e.g., to answer a question). 5 Points

Answer:

I decided to use langchain.llms to use OpenAI LLM models. I am using text-davinci-002 as simple model and text-davinci-003 as the more complex one to analyze differences in responses. In this section I just set up the simple model and ask it to tell me a joke to check that it works.

```
In [36]: gpt4_sector_mapping = {'3dtechnology': 'Tech', 'abtesting': 'Data Analytics and Management', 'accounting': 'Finan
```

```
In [17]: twenty_sectors = ['Consumer', 'Cybersecurity', 'Data Analytics and Management', 'Education',
                          'Energy', 'Entertainment', 'Environment', 'Finance and Banking', 'Food',
                          'Health Care', 'Legal Tech', 'Marketing', 'Media and Advertising', 'Other',
                          'Real Estate', 'Tech', 'Transportation', 'Travel and Hospitality',
                          'Construction and Manufacturing', 'Enterprise']

def question_builder(industry):
    return "The twenty sectors to choose from are the following: " + \
        ', '.join([str(sect) for sect in twenty_sectors]) + ".\n" + \
        "Chose a single sector that maps best the sub-industry: " + industry + "? " + \
        "Reply with no explanation, just the name of the chosen sector."

industries = ['3dtechnology', 'abtesting', 'accounting', 'adnetwork', 'adventuretravel', 'advertising',
              'agriculture', 'agtech', 'ai', 'airtransportation', 'analytics', 'android', 'app', 'apps',
              'art', 'assetmanagement', 'auctions', 'audio', 'augmentedreality', 'automotive', 'b2b',
              'banking', 'beauty', 'bigdata', 'billing', 'bioinformatics', 'biotechnology', 'bitcoin',
              'blockchain', 'brandmarketing', 'broadcasting', 'buildingmaterial', 'businessdevelopment',
              'cad', 'cannabis', 'careerplanning', 'charity', 'childeducation', 'cleantech', 'clinicaltrials',
              'cloud', 'cloudcomputing', 'clouddataservices', 'coaching', 'coding', 'coffee', 'collaboration',
              'collaborativeconsumption', 'commerciallending', 'computer', 'concerts', 'construction',
              'consulting', 'consumer', 'consumerapp', 'consumerelectronics', 'consumergoods',
              'consumerlending', 'consumertech', 'content', 'cooking', 'corporatetraining', 'coupons', 'crm',
              'crowdfunding', 'crowdsourcing', 'crypto', 'cryptocurrency', 'customerservice', 'cybersecurity',
              'database', 'datamanagement', 'datavisualization', 'dating', 'delivery', 'developerapis',
              'digitalentertainment', 'digitalmedia', 'digitalsignage', 'drone', 'ecommerce', 'edtech',
              'education', 'eldercare', 'elderly', 'elearning', 'electronichealthrecord', 'email',
              'emailmarketing', 'employment', 'energy', 'enterprise', 'enterprisear', 'enterprisesoftware',
              'enterprisetech', 'entertainment', 'europe', 'eventmanagement', 'events', 'fashion', 'finance',
              'financialservices', 'fintech', 'fintechsecurity', 'fitness', 'food', 'foodandbeverage',
              'foodbeverages', 'fooddelivery', 'foodprocessing', 'gaming', 'govtech', 'guides', 'hardware',
              'healthcare', 'healthdiagnostics', 'healthinsurance', 'hospitality', 'hr', 'insurance',
              'insurtech', 'intellectualproperty', 'internet', 'iot', 'it', 'languagelearning', 'legaltech',
              'locationbasedservices', 'logistics', 'machinelearning', 'manufacturing', 'marketing',
              'marketplaces', 'media', 'mediaandentertainment', 'messaging', 'mobileapps', 'mobilepayments',
              'music', 'networking', 'onlinelearning', 'opensource', 'other', 'payments', 'pitchdecktemplate',
              'productivity', 'proptech', 'realestate', 'recruiting', 'restaurants', 'retail', 'ridesharing',
              'saas', 'searchengine', 'security', 'sensor', 'socialimpact', 'socialmedia', 'software', 'sports',
              'supplychainmanagement', 'sustainability', 'tech', 'transportation', 'travel', 'travelevents',
              'venturecapital', 'video', 'virtualreality']
```

```
In [18]: from langchain.llms import OpenAI

import os
os.environ["OPENAI_API_KEY"] = "sk-295nnTV6pWQsVFXQ9VZST3B1bkFJFYTTTCJnVqR6GyMUJZ"
```

```
In [35]: llm_basic = OpenAI(model_name = "text-davinci-002", n = 2, best_of = 2)

llm_basic("tell me a joke").replace("\n", "")
```

Out[35]: 'Why did the chicken cross the road?To get to the other side!'

2. Run some basic experiments. Explain the experiments you ran and your observations (you may, for example, compare the results vs what you'd get on a public LLM like ChatGPT; e.g., if you're running a smaller model, what do you find?). 10 points.

Answer:

For my experiment I have a list of many industries and want to standardize them into 20 broader sectors that are given, using LLMs. Here I compare the results with text-davinci-002 and text-davinci-003, two OpenAI models that differ in size, the latter being much deeper than the former.

As ground truth I am using the mapping generated by gpt4, the most accurate LLM publicly available as of now.

I am showing both how well the LLMs can understand the question and provide a valid output (i.e. how many errors they produce) and how accurate they are in generating a similar sector distribution to gpt4 (shown both graphically and with the total error distance).

As expected, Davinci-002 sometimes fails to understand that it has to decide among a finite set of outputs that are given in the question and proceeds to provide an invalid answer. Davinci-003 also experiences this, but way less often (~1/4th of the time). Furthermore, the answers provided by Davinci-003 distribute across sectors more accurately (roughly twice as better as Davinci-002) with respect to the chosen ground truth, which was expected given the higher complexity of the language model.

```
In [33]: results = dict(zip(twenty_sectors, [[0, 0] for i in range(len(twenty_sectors))]))
basic_errors, advanced_errors = (0, 0)

llm_advanced = OpenAI(model_name = "text-davinci-003", n = 2, best_of = 2)
for industry in industries:
    basic_result = llm_basic(question_builder(industry)).replace("\n", "")
    advanced_result = llm_advanced(question_builder(industry)).replace("\n", "")
    try:
        tmp = results[basic_result]
        tmp[0] += 1
        results[basic_result] = tmp
    except:
        basic_errors += 1

    try:
        tmp = results[advanced_result]
        tmp[1] += 1
        results[advanced_result] = tmp
    except:
        advanced_errors += 1

print(results)

{'Consumer': [10, 7], 'Cybersecurity': [13, 4], 'Data Analytics and Management': [5, 8], 'Education': [4, 10],
'Energy': [0, 1], 'Entertainment': [12, 16], 'Environment': [1, 2], 'Finance and Banking': [13, 19], 'Food': [7,
8], 'Health Care': [11, 10], 'Legal Tech': [2, 3], 'Marketing': [6, 8], 'Media and Advertising': [3, 6], 'Other': [24, 13], 'Real Estate': [2, 2], 'Tech': [4, 21], 'Transportation': [4, 8], 'Travel and Hospitality': [2,
3], 'Construction and Manufacturing': [3, 3], 'Enterprise': [5, 8]}
```

```
In [44]: ground_truth_distribution = dict(zip(twenty_sectors, [0] * len(twenty_sectors)))
for k, v in gpt4_sector_mapping.items():
    ground_truth_distribution[v] += 1

print(ground_truth_distribution)

{'Consumer': 15, 'Cybersecurity': 3, 'Data Analytics and Management': 7, 'Education': 9, 'Energy': 1, 'Entertainment': 10, 'Environment': 4, 'Finance and Banking': 18, 'Food': 8, 'Health Care': 9, 'Legal Tech': 2, 'Marketing': 3, 'Media and Advertising': 10, 'Other': 7, 'Real Estate': 2, 'Tech': 34, 'Transportation': 6, 'Travel and Hospitality': 5, 'Construction and Manufacturing': 3, 'Enterprise': 13}
```

```
In [34]: print("Basic Errors: ", basic_errors, "\nAdvanced Errors: ", advanced_errors)
```

```
Basic Errors: 38
Advanced Errors: 9
```

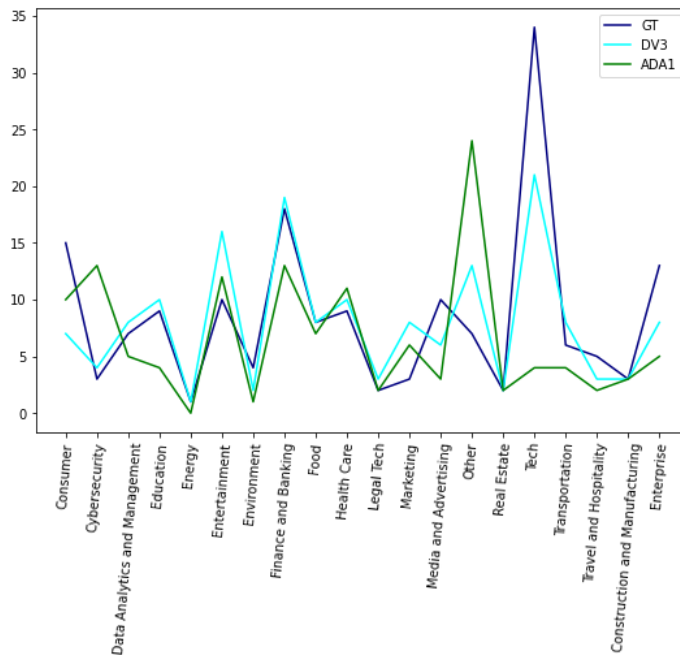
```
In [78]: # distributions:
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize = (9, 6))
basic_res = np.array([x[0] for x in results.values()])
advanced_res = np.array([x[1] for x in results.values()])
gt_res = np.array(list(ground_truth_distribution.values()))

ax.plot(gt_res, label='GT', color = "navy")
ax.plot(advanced_res, label = 'DV3', color = "cyan")
ax.plot(basic_res, label = 'ADA1', color = "green")

ax.set_xticks(np.arange(len(twenty_sectors)))
ax.set_xticklabels(twenty_sectors, rotation = 85)
ax.legend()
plt.show()

print("basic distance: ", np.sum(np.abs(basic_res - gt_res)))
print("advanced distance: ", np.sum(np.abs(advanced_res - gt_res)))
```



basic distance: 106
advanced distance: 59

3. Additional implementation: use your imagination here to get the model to do something interesting.. it could be as simple as prompt-tuning (e.g., if you're using a smaller model running locally, what would you do to improve performance). 20 points

Answer: Here I want to expand on the study by trying to influence the LLM's decision by providing more information in a creative way. Effectively I am telling the model question after question the partial distribution of its choices, and suggesting the model to try, when in doubt, to converge towards the actual distribution.

This practice can be quite important in real time scenarios when the distribution of the outputs is known or easily estimatable, for the given task. In this case, the general distribution of startups across sectors can be inferred through polls and statistical studies present in literature, but for the sake of simplicity I have decided to use the actual distribution of the ground truth.

Note that this is not a valid way from a statistical standpoint since the real distribution can massively differ from the one of the small subset that I take into consideration. Furthermore, in a real scenario, the ground truth is unknown, hence the distribution should be estimated from other sources, such as polls and studies as explained above.

```
In [92]: def advanced_question_builder(industry, curr_dist):
    return "The twenty sectors to choose from are the following: " + \
        ', '.join([str(sector) for sector in twenty_sectors]) + ". \n\n" + \
        "This is the current distribution of your choices: " + \
        ', '.join([(str(k) + " = " + str(v)) for (k, v) in curr_dist.items()]) + ". \n\n" + \
        "This is the target distribution: " + \
        ', '.join([(str(k) + " = " + str(v)) for (k, v) in ground_truth_distribution.items()]) + ". \n\n" + \
        "Chose a single sector that maps best the sub-industry: " + industry + "? " + \
        "Reply with no explanation, just the name of the chosen sector."
```

```
In [95]: basic_distribution = dict(zip(twenty_sectors, [0] * len(twenty_sectors)))
advanced_distribution = dict(zip(twenty_sectors, [0] * len(twenty_sectors)))

basic_errors, advanced_errors = (0, 0)

for industry in industries:
    basic_result = llm_basic(advanced_question_builder(industry, basic_distribution)).replace("\n", "")
    advanced_result = llm_advanced(advanced_question_builder(industry, advanced_distribution)).replace("\n", "")
    try:
        basic_distribution[basic_result] += 1
    except:
        basic_errors += 1
    try:
        advanced_distribution[advanced_result] += 1
    except:
        advanced_errors += 1

print("Basic Distribution: ", basic_distribution)
print("Advanced Distribution: ", advanced_distribution)
```

Basic Distribution: {'Consumer': 8, 'Cybersecurity': 3, 'Data Analytics and Management': 4, 'Education': 6, 'Energy': 0, 'Entertainment': 15, 'Environment': 1, 'Finance and Banking': 15, 'Food': 7, 'Health Care': 6, 'Legal Tech': 2, 'Marketing': 3, 'Media and Advertising': 4, 'Other': 3, 'Real Estate': 1, 'Tech': 44, 'Transportation': 6, 'Travel and Hospitality': 2, 'Construction and Manufacturing': 3, 'Enterprise': 9}

Advanced Distribution: {'Consumer': 6, 'Cybersecurity': 3, 'Data Analytics and Management': 5, 'Education': 9, 'Energy': 1, 'Entertainment': 13, 'Environment': 2, 'Finance and Banking': 16, 'Food': 8, 'Health Care': 11, 'Legal Tech': 2, 'Marketing': 5, 'Media and Advertising': 6, 'Other': 6, 'Real Estate': 2, 'Tech': 35, 'Transportation': 6, 'Travel and Hospitality': 6, 'Construction and Manufacturing': 3, 'Enterprise': 6}

4. Experiments and observations on what you implemented. Run some tests, explain your results and how they could be improved. 15 points.

```
In [96]: print("Basic Errors: ", basic_errors, "\nAdvanced Errors: ", advanced_errors)
```

Basic Errors: 27
Advanced Errors: 18

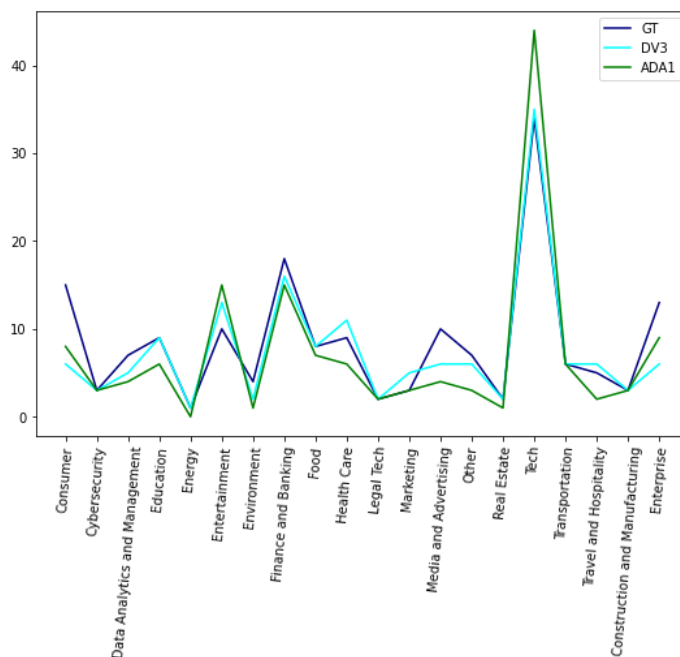
```
In [100]: # distributions:
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize = (9, 6))
basic_res = np.array([x for x in basic_distribution.values()])
advanced_res = np.array([x for x in advanced_distribution.values()])
gt_res = np.array(list(ground_truth_distribution.values()))

ax.plot(gt_res, label='GT', color = "navy")
ax.plot(advanced_res, label = 'DV3', color = "cyan")
ax.plot(basic_res, label = 'ADA1', color = "green")

ax.set_xticks(np.arange(len(twenty_sectors)))
ax.set_xticklabels(twenty_sectors, rotation = 85)
ax.legend()
plt.show()

print("basic distance: ", np.sum(np.abs(basic_res - gt_res)))
print("advanced distance: ", np.sum(np.abs(advanced_res - gt_res)))
```



basic distance: 57
advanced distance: 36

Answer:

I noticed that Davinci-002 benefits from the updates implemented in this experiment, reducing the invalid outputs by ~29%. On the other hand, Davinci-003 generates invalid responses twice the time. I am not exactly sure on how to interpret these results, it seems like a more complex prompt helps a simpler model to understand the requirements set in the question, while the deeper model gets confused by the overload of information.

From an accuracy standpoint instead, the experiment significantly improves both models' performance. Trying to force the responses to match the estimated distribution of the output (or in this case the true distribution) is a valid statistical method to improve the accuracy of a model, as it is shown by the massive improvements in accuracy achieved by both models.

With more time, it would be interesting to improve on this experiments implementing a more rigorous approach, through the following points:

- Find and use the estimated distribution instead of the real distribution of the ground truth.
- Perform more extensive prompt tuning merely on how the question is phrased, with the goal of minimizing the invalid outputs provided by both models.
- Experiment further with other models and tweaking hyperparameters of the current models used.
- Take into consideration other metrics alongside output-distribution accuracy and output validity, such as per-class accuracy.

In []: