University of Claude Bernard Lyon 1
Institut de Science Financière et d'Assurances **(I.S.F.A)**

Student project

Large Language Models applied to the insurance domain

Authors: Jérémy Liabeuf, Esther Lasly Gatoni, Mohamad Jalloul

Field of education: Master 1 Econometrics and Statistics

Supervisor: Mr.Couloumy

Year 2022-2023

Quick explanation of our subject

Large Language Models (LLMs) are special software programs typically created using the Transformer architecture and containing several parameters. They are also large language models used in insurance companies.
We will show a comparison of language models such as GPT-2, Bloom, and Llama, and identify the limitations of these large language models. Then we will train our model using GPT-2 on different sample results as a use case in insurance.

Summary

I) Introduction

1) What is a LLM?

a) Definition

Large Language Models (LLMs) are special software programs that computers can use to understand language. They're usually made using something called the Transformer architecture, and they're trained on massive amounts of text that people have written. By doing this, they learn how language works and can do things like come up with new words and sentences, or translate text from one language to another. Some of the most powerful LLMs have billions and billions of "parameters," which are just tiny pieces of information that help the computer understand language better. They work by trying to guess what word should come next in a sentence based on the words that came before it.

b) Why do we use it in insurance?

In the insurance field, LLMs can help simplify and automate processes, leading to better efficiency, reduced costs, and happier customers.

Large language models (LLMs) can be very useful for insurance companies in several ways. First, they can help them provide better customer service by assisting with inquiries and routing them to the appropriate service or agent for prompt resolution. Second, LLMs can help streamline claims processing by analyzing large amounts of data and improving efficiency. Finally, insurers can benefit from LLMs for risk assessment, as they can better predict risk by analyzing huge amounts of data. Overall, LLMs are a valuable tool for insurance companies, offering improved customer service, streamlined claims processing, and better risk assessment.

Let's dive into the math behind these large language models

2) Math behind LLMs

Large language models, are based on a type of neural network called the Transformer architecture. The math behind these models primarily involves linear algebra, probability theory, and calculus. Here's a detailed explanation of the key concepts and components involved in these models:

1. Tokenization
   Text data is transformed into numerical values called tokens by utilizing a pre-defined vocabulary. Each of these tokens corresponds to a word or sub-word present in the text. The tokenizer is an essential piece of technology utilized in language model training that helps machine learning systems better understand human language. Taking into account a fixed limit for the number of tokens, a subword tokenizer compresses letters into meaningful words utilizing statistical models to achieve an optimal compression rate. The tokenizer must be capable of compressing the entire

English language into a vocabulary of about 50,000 tokens. To perform this compression task, the tokenizer constructs cards one by one from token chains (a sequence of characters) to token IDs (a unique number). The tokenizer also includes punctuation tokens such as commas, periods, and spaces to ensure that every form of English language on the internet is symbolized effectively. Essentially, the tokenizer scrutinizes a string of text, sets it apart into token chains, and then converts it into a list of token IDs.

2. Embeddings
   To help a computer understand the meaning and relationships between words, we convert the words into a set of continuous numbers. This method of mapping words is done by using an integration matrix, which allows the computer to interpret the meaning behind the words. We can use a transformer like GPT-2 or Meta OPT to convert the word IDs into vector tokens. Depending on the model, the state vector size may vary, ranging from 768 to 4096. Essentially, the integration matrix can be thought of as a list of vector tokens, with each token corresponding to a particular word. In order to obtain the right symbolic vector, we simply use the token ID.

3. Positional encoding
   Since the Transformer architecture doesn't have any inherent understanding of the order of tokens, positional encoding is added to the input embeddings. This helps the model understand the position of each word in a sequence.
   Transformers use an intelligent positional coding, where each position or index is mapped to a vector. Therefore, the output of the positional coding layer is a matrix, where each row in the matrix represents a coded object in the sequence summed with its position information.
   The encoding is typically a sinusoidal function, which is added elementwise to the input embeddings.
   This sinusoidal function is given by sinus and cosinus functions of variable frequencies:

$$P(k, 2i) = \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{\frac{2i}{d}}}\right)$$

4. Self-attention mechanism
   In the Transformer architecture, there's this really interesting part called the self-attention mechanism. It basically helps the model to figure out which parts of the input sequence are more important and then it adjusts

them to create the final output. This is a really important feature in tasks that involve language processing because a word's meaning can change depending on its context in the sentence or the document. The Transformer uses a type of self-attention called scale scalar product attention to achieve this, which is integrated into its architecture. In this self-attention, there are three learnable matrices: Q, K, and V. They're computed by multiplying the input embeddings with their respective weight matrices (Wq, Wk, and Wv).

5. Scaled dot-product attention
   To calculate the attention scores between each pair of words in a sequence, we use the dot product of their query and key vectors. After that, we scale the result by the square root of the dimension of the key vectors (dk). These scores are then run through the softmax function to obtain a probability distribution. We use this distribution to weight the value vectors, and then sum the weighted value vectors for the final output..
   To calculate the attention of the scaled point product this is done in 4 steps:

   - Calculate the alignment scores by multiplying all the queries packed in the Q matrix, with the keys in the K matrix.
   - Scale of each alignment score by $\frac{1}{\sqrt{dk}}$
   - Another escalation process by applying a maximum operation in order to obtain a set of weights.
   - Finally, apply the weight resulting from the values of the matrix V of dimension.

6. Multi-head attention
   The Transformer architecture uses multiple attention heads instead of a single one for capturing different relationships between words. To obtain the final multi-head attention output, linear projections of queries, keys, and values are calculated with weight matrices. Then, a unique attention feature is applied to each head, which involves multiplying the queries and key matrices, scaling and softmax operations, and weighting the value matrix. The resulting outputs from each head are concatenated and passed through a linear layer to generate the final result.

7. Feed-forward neural network
   The output of the multi-head attention goes through a feed-forward neural network that consists of two linear layers separated by an activation function, which is further processed by position. However, the standard MLP and its derivatives have a drawback in that there is a strong relationship between the connection graph and the weight network. The graph is

only used to allow communication between neurons, but the weights are connected to the idea of control structure, which helps in modifying the local computation of each neuron during training.

8. Residual connections and layer normalization
   In the Transformer architecture, there are two features that are used for training and performance enhancement. Residual connections and layer normalization are used to mitigate the problem of vanishing gradient and enhance the training process. For each layer in the model, normalization of the output is done, which helps with stable learning. Also, a residual connection is added, which allows the input to be directly passed to the output. It helps in making the model learn which parts of input are crucial. By repeating these components several times, a deep neural network is constructed. It allows the model to process extended text sequences and create better outputs for various language tasks like query answering, text generation, and translation.

9. Stacking layers
   The Transformer architecture has various encoder and decoder layers that are stacked on top of each other. The encoder layers have a multi-head attention mechanism, followed by a position-wise FFN. On the other hand, each decoder layer includes an extra multi-head attention layer that focuses on the encoder's output.

10. Output layer
    The final output from the top decoder layer goes through a linear layer and a softmax layer, resulting in a probability distribution over the vocabulary. The word with the highest probability is then selected as the prediction.

Following our discussion of the foundations and capabilities of large language models, a comparative examination of these models is required to better appreciate their strengths and drawbacks. In the following part, we will go over the comparative advantages, disadvantages, and performance constraints of various large language models in further detail.

II) Comparison and limits of LLMs

Table 1: Comparison of GPT-2, Bloom, and Llama

Aspect	**GPT-2**	**Bloom**	**Llama**
Model Architecture	Large-scale Transformer-based	Efficient Transformer	Flexible choice
Pre-trained Knowledge	Yes	Yes	N/A
Fine-tuning Capabilities	Yes	Yes	Yes
Inference Speed **Memory Requirements**	Slower Higher	Faster Lower	Depends Depends
Community and Support	OpenAI	Hugging Face	Hugging Face
Recommendation for Insurance	Bloom		GPT-2

1) GPT-2 vs Bloom

Have you heard of Hugging Face Bloom and OpenAI GPT-2? They are both top-of-the-line models used in natural language processing, but they have their own unique features in terms of how they're built, what they can do, and where they are used. Let's take a detailed look at how these two models compare.

a) Model Architecture

1. Hugging Face Bloom:
   Bloom is a really cool type of Transformer that was made by Hugging Face. It helps the Transformer use less memory and be faster, but still work just as well. They did this by using layer pruning, weight sharing, and quantization. The people who made BLOOM trained it using a combination of two things: Megatron-DeepSpeed and Megatron-LM. Megatron-DeepSpeed is a library that helps people with deep learning make their work happen faster, and Megatron-LM is a transformer model idea that was made by an awesome team at NVIDIA. The people on the DeepSpeed team used something called ZeRO partitioning and pipeline parallelism to implement BLOOM.

2. OpenAI GPT-2:
   GPT-2, which stands for Generative Pre-trained Transformer 2, was created by OpenAI as a language model. GPT-2 has different sizes from 117M to 1.5B parameters. The larger models have better performance but require more computational resources. This model was a precursor to GPT-3 and GPT-4, which are more advanced.

b) Pre-trained Knowledge

Both Bloom and GPT-2 are trained on lots of texts to learn how to use words, understand grammar, and some real facts. However, what they are trained on and how much they learn can be different.

c) Fine-tuning Capabilities

Fine-tuning Bloom and GPT-2 models on specific industry datasets can help them perform better on tasks like insurance. This is achieved by refining the models' general pre-trained knowledge to fit the particularities of the industry or use-case at hand.

d) Inference Speed and Memory Requirements

Compared to older Transformer models like the GPT-2, Bloom is intended to be more memory- and inference-speed-efficient. When models are deployed in production, especially in contexts with limited resources or when real-time answers are required, this can be a substantial advantage.

e) Community and Support

Hugging Face and OpenAI both have strong communities and extensive resources, including tutorials, example projects, and pre-trained models, which can be helpful when customizing and implementing these models for specific tasks.

2) Llama vs GPT-2

a) Architecture:

An open-source framework called Llama (Language Model as a Service) is used to develop and distribute unique language models. It is based on Hugging Face's Transformers library and works with a number of architectures, including BERT, RoBERTa, and GPT-2. Because of this, there are more options for architecture.

b) Data requirements:

You can use Llama to refine pre-trained models on your particular dataset or create a bespoke model from scratch. You can pick an appropriate architecture and train or fine-tune the model on your domain-specific dataset for insurance applications.

c) Adaptability:

Llama is a good option for a variety of applications, including insurance, because it can be customized for training and deployment. Depending on the training data and architecture selected, it can be used for tasks including text categorization, named entity recognition, and sentiment analysis.

Both GPT-2 and Llama can be used for custom training in the insurance industry. Choosing the best option for your specific use case will depend on your requirements and available resources.

Key points to consider when making your decision

GPT-2:

- Powerful pre-trained model with a larger number of parameters.
- Highly adaptable and can be fine-tuned for various tasks, such as text classification, sentiment analysis, and summarization.
- More resource-intensive due to its model size.

Llama:

- Flexible choice of architectures, as it is built on top of Hugging Face's Transformers library.
- Designed specifically for custom training and deployment.
- Supports a variety of tasks, depending on the chosen architecture and training data.

GPT-2 may be a better option if you have a sizable dataset and sufficient computing capacity because it is a potent pre-trained model that can be customized for different jobs. It has been demonstrated to be effective for a variety of natural language processing tasks, making it a good option for applications involving insurance.

Llama, on the other hand, might be a better alternative if you value flexibility in architectural selection and need a solution that is intended expressly for individualized training and deployment. With Llama, you may select the architecture that is best for your use case and train or improve the model using data that is specific to your domain.

3) Limits of LLM and how to overcome them

Large language models can have huge implications for the future of language processing. These models need to understand context and word meaning and generate accurate and complex responses to user queries, and LLMs perform a wide range of tasks.

However, they have certain limitations that can affect their performance and usefulness, especially in specialized areas such as insurance.

1. Generalization and lack of domain-specific knowledge.
2. Data quality and bias.
3. Lack of interpretability and explainability.
4. Ethical considerations.
5. Computational resources.

Let's dive into each of these limitations one by one.

Here, we will focus on the first limitation mentioned earlier: generalization and lack of domain-specific knowledge.

a) Generalization and lack of domain-specific knowledge

Large language models are trained on large and diverse data sets, allowing them to learn and understand a variety of topics, including common sense. However, these models may not have comprehensive or accurate knowledge in specific areas such as insurance, advanced scientific topics, or legal issues. For example, an LLM such as GPT-2 or Bloom's can generate a general description of how car insurance works, but may struggle with more specialized concepts such as actuarial tables, risk assessment algorithms, or reinsurance principles.

> **Example:**
>
> Imagine you ask an LLM a question related to the insurance domain:
>
> *Question:* Can you explain the difference between experience rating and community rating in health insurance premium pricing?
>
> An LLM with a lack of domain-specific knowledge might provide a vague or incomplete answer.
>
> *Answer:* Experience rating and community rating are two methods used to determine health insurance premiums. Experience rating considers the individual's health history, while community rating takes into account the health status of the entire community.

Although this response is partially correct, it doesn't show a clear understanding of the subject matter as a whole. When insurance companies calculate premiums for individuals or groups, they consider their claim history. However, when it comes to Community Ratings, premiums are based on the average cost of care for the entire community, regardless of the age, gender, and health status of the individuals. To overcome this drawback, LLM can be improved by using organized datasets. This approach helps the model gain more precise knowledge and come up with better responses. Nevertheless, maintaining the data set's quality is essential to avoid inaccuracies. Moving on to the second drawback, let's discuss data quality and bias in more detail.

b) Data quality and bias

Large language models (LLM) are trained using large amounts of textual data collected from various sources. The quality and variety of the training data have a direct impact on the model's performance. If there are biases or inaccuracies in the training data, the results can be problematic, leading to erroneous outputs, particularly in fields such as insurance.

Various methods can be employed to reduce bias in LLM, including data preprocessing to review and preprocess the training data to identify and remove biased content, post-mortem analysis to analyze the output produced by the model to identify and correct outliers, and algorithmic fairness interventions using techniques such as adversarial training or fairness-aware machine learning algorithms to reduce the model output bias.

Therefore, ongoing monitoring and evaluation of LLM results are essential to maintain fairness, inclusiveness, and accuracy, particularly in industries such as insurance, where biased decisions can significantly affect individuals or groups.

lets move on to our third main limitation.

c) Lack of interpretability and explainability

Large language models are really complicated pieces of technology with millions or billions of parameters that are used to make decisions. However, it can be hard to understand why they choose certain outcomes. Essentially, these models are like magic - they work, but we don't know how. In certain fields, such as insurance, it's important for people to understand why certain decisions are made since there can be serious consequences for customers, regulators, and business owners. Unfortunately, it's difficult to explain these decisions when using large language models.

An insurance company may use a system to determine the premium an individual will pay. This system utilizes a Risk Score, which can result in higher premiums. However, if the decision is questioned, the company may struggle to provide a clear explanation due to the complexity of the system. To make it simpler, organizations can use more understandable models like decision trees and logistic regression. They can also use an approach where human experts review and provide explanations for decisions made by the system. By combining these approaches with the original system, organizations can balance the power of the system while maintaining transparency and accountability.

Lets move on to our forth limitation.

d) Ethical considerations.

The use of big computer models in different areas, such as the insurance industry, raises ethical issues that must be taken into account. Some of the principal ethical problems that come with using these models include:

1. Unintentional prejudice and discrimination: As we have mentioned before, these models can unknowingly learn and transmit biases that are present in the data used to train them. This can lead to results that are unjust, offensive, or discriminatory. In the context of insurance, biased decisions can seriously impact people's ability to get coverage or how much they have to pay in premiums.

2. Privacy concerns: These models can also inadvertently store and reveal personal information and other sensitive data that are present in the data used to train them. In the context of insurance, sharing sensitive customer information or trade secrets can have serious legal and financial consequences.

3. Misleading information and manipulation: These models can create text that is so convincing that it may be mistakenly believed to be accurate, even if it is completely false. In the insurance industry, misinformation can damage a company's reputation, confuse customers, or create confusion in the market.

Example:

Suppose an insurance company uses an LLM to generate a personalized policy recommendations for potential customers. If the LLM has learned of bias or inaccuracies in its data, it may produce recommendations, which could lead clients to purchase inadequate coverage or be unfairly charged higher premiums.

Organizations can take certain steps to address ethical concerns regarding machine learning models. For instance, they can invest in techniques to detect and decrease bias, as well as use algorithms to minimize bias in model outputs. Other strategies include anonymizing and pre-processing training data to remove private information and protecting the privacy of individuals in datasets, implementing content filtering mechanisms to avoid improper text generation, and continuously evaluating the outcomes to ensure ethical standards are met. Moreover, it is recommended that organizations clearly explain the significant drawbacks and risks of machine learning models to stakeholders and establish procedures to guarantee accountability for model decisions.

And finally let's explain the last limitation of LLMs.

e) Computational resources

Training and adjusting large language models is a process that requires a lot of computer power and memory. These models are made up of millions or even billions of parameters, which makes it take a long time to train, adjust, and use them. This can be hard for small organizations, startups, or people with few resources, particularly when creating custom models for specific applications, like the insurance industry..

Additionally, the energy consumption associated with training and fine-tuning LLMs raises environmental concerns, as the carbon footprint of training these models can be substantial.

Example:

A small insurance company needs help processing insurance claims with technology. However, they may not have the budget or technical skills to use complex models such as GPT-2 or Bloom. Because of this, they may struggle to fully benefit from advanced language models for their specific needs.

To deal with the problem of not having enough computer power, organizations can consider a variety of methods. Firstly, they can use smaller and more efficient models that won't need as many resources to train, fine-tune, and operate. Additionally, organizations can save time and resources by using pre-trained models and fine-tuning them to their specific domain. Cloud-based machine learning platforms and services are another option. Organizations can use these platforms and services which gives access to the required computational resources on a pay-as-you-go basis. Lastly, smaller businesses can partner with research institutions, or technology partners if they don't have the required resources or expertise to develop and deploy LLMs.

For our insurance training, we chose GPT-2 over Llama and Bloom because it has superior pre-trained models, is versatile for fine-tuning, and has shown success in various natural language processing applications. Even though Llama offers more flexibility and Bloom is more efficient, we found that GPT-2's performance and adaptability make it a better choice for insurance-related tasks.

Now that we've gone into the complexities of huge language models and their limits in Section 2, we can apply what we've learned to create a specific solution for the insurance sector. In Section 3, we will focus on developing a custom-trained model for insurance applications, using the strength of GPT-2 while reducing its limits to create a highly effective tool tailored to the insurance sector's specific demands.

III) Our custom-train model using GPT-2

In this part, we are going to make an example with our custom-train model based on GPT-2 to see if the model can be used to make useful things. We coded the model with the help of the Python language.

1) Base of our project

a) The workspace

Firstly, we needed a workspace to train our model. The problem was that LLMs use a lot of power especially with a graphic card. Our own computers weren't powerful enought to run the model, so we created a Google Colab notebook. The main advantage is that we can use the more powerful GPU (Graphic card) the notebook provides to use it on our model.

The specifications of the notebook are the following:

- Processor: Intel Xeon 2.2 GHZ
- Graphic card: Nvidia Tesla T4 16 GB
- Memory: 12 GB
- Disk memory: 78 GB

The second thing is that we don't need to install extra softwares like ipython because everything is already installed. The link for the notebook is included in the reference tab.

b) The model

To fine-tune GPT-2, we chose GPT-2 simple. It is a Python package that makes it easy to fine-tune and generate text from GPT-2, and that works best with the "small" version of 124M parameters. It also allows to generate text to a file for easy curation, and to use prefixes to force the text to start with a given phrase. `gpt-2-simple` allows you to generate texts in parallel by setting a `batch_size` that is divisible into `nsamples`, resulting in much faster generation. The link is available in the references.

2) Starting point

In this section, we are begin to talk about the Python's commands we use to define our project. We are going to explain in precision each command.

a) Installation

First we need to install the model. We are going to use the following command:

```
!pip install gpt-2-simple
```

This command is used to install the `gpt-2-simple` Python package.

- The `!` at the beginning is used to run a shell command from Google Colab.
- The `pip` command is a package manager for Python that allows you to install and manage packages.
- The `install` arguments tells `pip` to install the specified package, in this case `gpt-2-simple`.

After that, we are going to rename `gpt-2-simple` to `gpt2` with next the command to simplify the following steps :

```
import gpt_2_simple as gpt2
```

Where `import` means that it imports the `gpt-2-simple` module and gives it the alias `gpt2`.

Next, we will download the model:

```
gpt2.download_gpt2(model_name="124M")
```

This command downloads the GPT-2 model with 124 million parameters into our Google Colab notebook. This is a pre-trained language model that can be tuned for a specific task or used to generate text. This is why we have `model_name="124M"`. The pre-trained data weights around 498 MB.

The data we have downloaded are stored inside the `\content` folder of our Google Colab. There are 2 folders inside it:

- models: This is where the GPT-2 pre-trained model is stored.
- sample_data: This is some sample of data that the model has downloaded (we can ignore it).

b) Sample to fine-tune

Now that we have downloaded the GPT-2 simple model, we are going to initialize it.

We select a text file (.pdf, .txt, etc...), convert it to a .txt format because GPT-2 only supports this format for fine-tuning. Then, we upload it to the Google Colab workspace and we run the following command:

```
file_name = "Example.txt"
```

This code assigns the text name in the workspace `Example.txt` to a variable called `file_name`. We do this to make it easier to use this variable for the next section.

3) Fine-tuning our sample

a) Initialization

In this third part, we are going to initialize the fine-tuning model with the following set of commands:

```
from tensorflow.python.framework import ops
ops.reset_default_graph()
sess = gpt2.start_tf_sess()
gpt2.finetune(sess,
              dataset=file_name,
              model_name="124M",
              steps="",
              restore_from="fresh",
              run_name="example",
              print_every="",
              sample_every="",
              save_every="",
              only_train_transformer_layers = True,
                    accumulate_gradients = 1,
              )
```

The first two lines were added because we had an error, it will be explained later in the part where we talked about the issues.

```
from tensorflow.python.framework import ops
```

This line imports the `ops` module from the `tensorflow.python.framework` package. The `ops` module includes implementations of classes like `tf.Graph`, `tf.Tensor` and `tf.Operation`.

```
ops.reset_default_graph()
```

This line resets the default computational graph used by TensorFlow. This is useful if we want to create a new graph from scratch.

```
sess = gpt2.start_tf_sess()
```

This line creates a new TensorFlow session using the `start_tf_sess` function from the `gpt2` module and assigns it to the variable `sess`.

We are now going to see what the `gpt2.finetune` function is going to do:

```
gpt2.finetune(sess,...)
```

The first line fine-tunes the GPT-2 model on a specific dataset using the `finetune` function from the `gpt2` module with the sess variable.

The following parameters we used for this function are:

- `dataset=file_name`: This specifies the dataset to use for fine-tuning. In this case, it is set to the precedent value `file_name`, which should contains the name of a text file.
- `model_name="124"`: This specifies which version of the GPT-2 model to use. In this case, it is set to `"124M"`, which corresponds to the version of the GPT-2 model with 124 million parameters.
- `steps=""`: This specifies how many steps (iterations) to perform during fine-tuning. The higher the number of steps is, the better the quality of the model is.
- `restore_from="fresh"`: This specifies whether to start fine-tuning from scratch or to restore from a previous checkpoint. In this case, it is set to `"fresh"`, which means that fine-tuning will start from scratch.
- `run_name="example"`: This specifies a name for this fine-tuning run. It can be used to distinguish between fine-tuning runs.
- `print_every=""`: This specifies how often (in terms of steps) to print progress information during fine-tuning.
- `sample_every=""`: This specifies how often (again in terms of steps) to generate text samples during fine-tuning. It means that it will print every number of steps you specified.
- `save_every =""`: This specifies how often to save a checkpoint during fine-tuning. You can run a model again with a specified checkpoint.
- `only_train_transformer = True`: This specifies whether only transformer layers should be trained during fine-tuning or not.
- `accumulate_gradients = 1`: This specifies how many gradients should be accumulated before updating model parameters.

We can execute the commands after setting the parameters. The execution time will be longer if the sample is larger or if we use powerful parameters like `steps=1000` for instance.

Afterward, we are going to have our fine-tuned checkpoint. We can generate now a random sample with this command:

```
gpt2.generate(sess, run_name="example")
```

Where `example` is our checkpoint we trained previously.

b) Generate a text with our checkpoint

We can add those following parameters to generate a better and a more custom sample within the `gpt2.generate` function:

```
gpt2.generate(sess,
              run_name="example"
              length="",
              temperature="",
              prefix="",
              nsamples="",
              batch_size="",
              top_k= ""
              )
```

- `gpt2.generate(sess`: This line calls the `generate` function of the `gpt2` object with `sess` as the first argument. `sess` is the previous TensorFlow session that contains information about our trained model.
- `run_name="example"`: The checkpoint you want to use for generating sample.
- `lenght=""`: This line sets the maximum length of the generated text in number of tokens. The maximum number you can generate with GPT-2 is about 1023 tokens.
- `temperature=""`: This line sets the temperature for sampling during text generation. A higher temperature can result in more diverse but less coherent results. A lower temperature can result in less diverse but more coherent and repetitive results. The default value of GPT-2 is 1.

- `prefix=""`: This line sets the prefix for the generated text. The model will start generating text with the context from this prefix.
- `nsample=""`: This line sets the number of text samples to generate, if you set this line to 10 for example, you will get 10 lines with different outputs. The default value sets by GPT-2 is 1.
- `batch_size=""`: This line sets the batch size for text generation. This means that the model will generate a number of samples given at a time until the total number of samples specified by `nsamples` is reached.
- `top_k=`‘“’: This line chooses how many possible words to look at when picking the next word. It will only look at the `top_k` most likely words for the next one.

We can use the model to create a new text and check our result, then test different samples to finish our analysis.

4) Results of the different samples

In this part, we will look at different results with the first chapter of the first Harry Potter book, the whole book of the first Harry Potter (to test our model's limits) and a financial report about a company called Workiva Inc. that is partly related to the insurance domain.

a) Sample 1: First chapter of Harry Potter and the Sorcerer's stone

We picked this example because Harry Potter's books are often used for fine-tuning test models, especially the first one. Harry Potter and the Sorcerer's stone is a fantasy book from 1997 by the author JK.Rowling (all rights belong to the original author of this book).

Firstly, we are going to import the text sample with this command:

```
file_name = "HP-GPT2.txt"
```

Next, we set those commands with the following parameters like this:

```
gpt2.finetune(sess,
              dataset=file_name,
              model_name="124M",
              steps="100",
              restore_from="fresh",
              run_name="run1",
              print_every="10",
              sample_every="10",
              save_every="10",
              only_train_transformer_layers = True,
                    accumulate_gradients = 1,
              )
```

The model took about 3 minutes to create with those parameters. Next, we will use these parameters to make a sample:

```
gpt2.generate(sess,
              length=200,
              run_name="run1",
              temperature=0.7,
              prefix="Harry potter is in danger",
              nsamples=10,
              batch_size=10,
              top_k= 10
              )
```

(We will adjust temperature to check if the text changes in content)

We chose the temperature to be 0.7 because it balances creativity and fidelity and it is the default recommendation. The sample was generated in about 7 seconds, which is quicker than the part when we trained our model.

Now for the analysis, we got a generated text with 200 tokens (The text is link in a github collection of our results in the biography table). When we set the temperature to 0.7, we get a fairly coherent text in terms of form. However, we can already see that there are repetitions of special characters such as "-" for example, or the repetition of certain words or phrases:

```
"Professor McGonagall opened her mouth, changed her mind,
    swallowed, and said, Yes, youd be certain, said Dumbledore.

Professor McGonagall opened her mouth, changed her mind,
    swallowed, and said, Yes, youd be certain"
```

In terms of coherence, GPT-2 has trouble maintaining context. For example, in a paragraph, the text tells us that the hero goes to prison because of a theft committed by himself, which has no connection with the first chapter of this book.

When we set the temperature to 0.2 this time, the first thing that catches the eye is the repetitive presence of the same paragraph:

```
"Professor McGonagall pulled out a lace handkerchief and dabbed
    at her eyes beneath her spectacles. Dumbledore gave a great
    sniff as he took a golden watch from his pocket and examined it
    . It was a very odd watch. It had twelve hands but no numbers
    ; instead, little planets were moving around the edge. It
    made no sense to Dumbledore"
```

As the model is limited in originality and the text is not very long, it is unable to choose another paragraph so it is limited to repeating another sample that comes out. Otherwise, the generated text relies heavily on the original, even if the logic behind is not present.

b) Sample 2: The entire book of Harry Potter and the Sorcerer's stone

We will use the whole book this time because we wanted to test if the model can handle big text data. We will keep the same parameters as the previous generation. The only thing we will change is this line:

```
file_name = "HPFull-GPT2.txt"
```

This is because we use another text sample which contains the full book.

```
run_name="run2",
```

We use another run name because the first one was already chosen from our previous generation.

```
gpt2.generate(sess,
              lenght=200,
              run_name="run2",
              [...]
              )
```

And also this one.

The training session took around 3 minutes, so we can conclude that the number of tokens from the text doesn't affect much the training time.

For generating a new sample, we are going to also use the previous parameters with the same `prefix="Harry potter is in danger"`.

We will just do a quick analysis to see if GPT-2 has any issues with the interpretation of large text corpora. When we set the temperature to 0.7, we observe on one hand that the model generates repetitions in some cases, like with the character `""` for example. The text appears coherent overall even if the generated story does not have a common thread. However, when the temperature is set to 0.2, the repetitions come back as in sample 1. We can see it here in particular:

```
"I’m not going to Hogwarts, I’m not going to Hogwarts, I’m not
    going to Hogwarts, I’m not going to Hogwarts, I’m not going
    to Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts, I’m not going to
    Hogwarts, I’m not going to Hogwarts,"
```

Or here:

```
"He’s in trouble, Harry," said Ron. "He’s got a wand, he’s got a
    curse, he’s got a curse, he’s got a curse, he’s got a curse,
    he’s got a curse, he’s got a curse, he’s got a curse, he’s
    got a curse, he’s got a curse, he’s got a curse, he’s got a
    curse, he’s got a curse, he’s got a curse, he’s got a curse,
    he’s got a curse, he’s got a curse, he’s got a curse, he’s
    got a curse, he’s got a curse, he’s got a curse, he’s got a
    curse, he’s got a curse, he’s got a curse, he’s got a curse,
    he’s got a curse, he’s got a curse, he’s got a curse, he’s
    got a curse, he’s got a curse, he’s got a curse"
```

The model seems quite confused. It appears that it is unable to generate new sentences and is content with repeating the same ones in a loop.

In conclusion, GPT-2 doesn’t seems to have difficulty to generate new samples from a larger text document.

With this successful "stress-test", we can now analyse the main aspect of our subject: a financial report which is linked to insurance.

c) Sample 3: Workiva Inc.'s financial report from 2021

The main goal in this part is to see if people can make predictions with LLMs such as GPT-2.

This is a financial report from a firm named Workiva Inc. which is dated from 2021. We use it because it has multiple pros:

- Good and trustworthy data: Workiva Inc.'s financial reporting software has correct and current financial data (at the time of this project is made), which makes it a good dataset for fine-tuning GPT-2.
- Knowledge about the industry: A financial report from Workiva Inc. would have useful information and insights about the financial industry, which could help GPT-2 make better predictions about insurance data .
- Easy collaboration and data sharing: Workiva Inc.'s platform lets people work together and share data easily, which can be helpful when working with GPT-2 models .
- Correctness and confidence: Workiva Inc.'s platform connects financial data from different sources, lowering human error and increasing data reliability. This can lead to better predictions from the GPT-2 model.

But also couple of cons:

- Narrow range: A financial report from Workiva Inc. would have useful information about the financial industry, but it might not include all parts of insurance data. The model might need more fine-tuning with more varied and complete datasets (if you need to make accurate predictions).
- Old data: Using a financial report from 2021 might not give the most current information for making predictions about insurance data. It's important to think about the importance and freshness of the data when fine-tuning the GPT-2 model.
- Data from others: Workiva Inc.'s financial reports may have information from other sources (like Deloitte tool for example), which could affect the correctness and dependability of the data investor. It's important to check the quality and trustworthiness of the data before using it for fine-tuning the GPT-2 model.

Presentation of Workiva Inc. and what is its link to insurance:

Workiva Inc. is a cloud-based platform that transforms the way people manage and report data, with a mission to power transparent reporting for a better world. The platform is used by thousands of organizations worldwide, including 85 percent of the Fortune 500, to streamline processes, connect data and teams, and simplify complex work.

In the insurance domain, Workiva Inc. streamlines complex insurance reporting, including statutory and financial reporting, risk management, and regulatory compliance across all lines of business. The platform can be used for financial and insurance statutory reporting, Long Duration Targeted Improvements (LDTI), actuarial opinions or actuarial memorandums, Model Audit Rule (MAR) compliance, and IFRS requirements.

By connecting data, documents, and teams, Workiva Inc. simplifies compliance with regulatory requirements such as Solvency II and ORSA, and can be used to produce actuarial opinions or an actuarial memorandum and manage LDTI.

The original report was in .pdf format, so we scrap the text inside it manually (there were a couple of graphics inside, but we didn't need them). The .txt file is named Finance-GPT2.txt.

We use the following parameters to set up the model:

```
from tensorflow.python.framework import ops
ops.reset_default_graph()
sess = gpt2.start_tf_sess()
gpt2.finetune(sess,
              dataset=file_name,
              model_name="124M",
              steps="1000",
              restore_from="fresh",
              run_name="run3",
              print_every="10",
              sample_every="10",
              save_every="10",
              only_train_transformer_layers = True,
                    accumulate_gradients = 1,
              )
```

We have `steps=1000` because the text has 695 lines in total and it will generate a better sample using more steps (and take much longer time to generate).

It took about 30 minutes to train the text, which is very long but we have to keep in mind that we are limited by the hardware, so it's normal for our case. Next, we will create the text with these parameters:

```
gpt2.generate(sess,
              run_name="run3",
              length="1023",
              temperature="",
              prefix="Prediction for 2023",
              nsamples="3",
              batch_size="3",
              top_k="40"
              )
```

We choose `length=1023` because we wanted to the see what the maximum number of tokens we can get per sample. We are going to generate 3 different ones : one with `temperature=1`, one with `temperature=0.2` and one with `temperature=0.5` (we generate 3 different samples within the sample, this is why we have `nsamples=3`):

- When we set the temperature to 1, the text is quite faithful to the original one. It even came to complete one of the main titles by adding "- 2025". Indeed, at this temperature, the text has a great deal of freedom to form new sentences and new words. However, from a financial and scientific point of view, this text does not really have any value because it only repeats absurd words. There are also certain terms that are misinterpreted as squares, for example. We can also see that GPT-2 has generated the beginnings of a list, particularly in the second paragraph from 7 to 10. We also have some <|endoftext|> that appear suddenly at certain points.

- When we set the temperature to 0.2, we can see that the text is much more constrained than when we set the temperature to 1. There are also many repetitions because GPT-2 is more constrained in the choice of words, especially with words such as 'We have not been' followed by pre-fabricated sentences. This is particularly the case here with sentences where it becomes confusing by repeating the same thing:

```
"Stock Performance Graph. The following shall not be
    deemed incorporated by reference into any of our
    other filings under the Exchange Act or the
    Securities Act.
Stock Performance Graph. The following shall not be
    deemed incorporated by reference into any of our
    other filings under the Exchange Act or the
    Securities Act.
Stock Performance Graph. The following shall not be
    deemed incorporated by reference into any of our
    other filings under the Exchange Act or the
    Securities Act.
[...]"
```

It also uses lists repeatedly, which shows that it closely follows the original document that also has lists. It again increases the title list like this:

```
"9. Fair Value of Financial Instruments
[...]
10. Fair Value of Financial Instruments"
```

Otherwise, it has generated a coherent text in form but without a serious meaning in substance.

- Finally, when we set the temperature to 0.5, we can see that there are no major drawbacks. The text is quite readable and coherent in form. There are still some repetitions, but they are far less noticeable than when the temperature is set to 0.2 or 1. This temperature choice could be ideal for a mix of both, on a one side, to make the text more creative, and on the other side, to inspire it from the original text.

With these 3 predefined temperatures, we can see that we have quite different results. If the temperature is close to 1, then the model will generate a more creative sample while if the temperature is close to 0, then the model will generate text close to the original one, therefore more limited. Note that we can also adjust other parameters such as `top_k` to have different results but we are staying in a simple case study.

We can summarize our analysis into the following table:

Table 2: Advantages and Disadvantages

Advantages	**Disadvantages**
- Easy to install	- Too repetitive when the temperature is low
- Can be run on a local machine	- Not very coherent
- Can analyse huge chunks of text	- Slow when you have a high number of steps to generate sample
- Good to generate creative content	- Outdated compared to other LLMs
- Lightweight	- Not good for professional stuff (like prevision)
- Privacy concerns	- Can require a lot of power for other parameters
	- Hallucinate words and sentences

Although GPT-2 is a good language model, it is not the best one available on the market.

You can see the final samples for each three at this address:
`https://github.com/AverageCoder69/TER_project_ISFA`

5) Issues with the process

At first, we selected the model with 365 million parameters because more parameters make the model more accurate and effective.

But we had a problem with it. In fact, `gpt-2-simple` was not optimized for this kind of parameters and secondly, the model took a lot of video ram and threw this error:

```
"ValueError: Variable model/wpe already exists, disallowed.
Did you mean to set reuse=True or reuse=tf.AUTO_REUSE
in VarScope?"
```

This error is occurring because the variable `"model/wpe"` already exists and cannot be created again. There are two solutions against this problem, either set `"reuse=True"` or `"reuse=tf.AUTO_REUSE"` in the variable scope.

But none of those solutions work for us. We decided to use these following lines of commands to correct the problem:

```
from tensorflow.python.framework import ops
ops.reset_default_graph()
sess = gpt2.start_tf_sess()
```

As we said, the first line imports the `ops` module from the `tensorflow.python.framework` package. The `ops` module provides a collection of functions that are used to construct TensorFlow graphs. The second line resets the default TensorFlow graph. This is useful when working with multiple graphs in the same program or notebook. The third line starts a new TensorFlow session using the `gpt2.start_tf_sess()` function. The gpt2 module provides an implementation of the GPT-2 language generation model.

IV) Conclusion

Our project aimed to test the possibility of using a pre-trained LLM, namely GPT-2, to create new samples that were fine-tuned with existing data. We managed to produce a "prediction" from a financial report about the insurance domain.

But it was not very good. There were many false statements, the text sometimes lacked coherence and there were also some repetitions. LLMs can be very useful but also can be ineffective for some tasks.

It is important to note that while GPT-2 was a competent LLM when it was launched in 2019, it is now considered outdated compared to other LLMs such as GPT-3 and GPT-4. However, these newer models require a lot of computational power to run since they have significantly more parameters than GPT-2, with GPT-3 having around 175 billion parameters, which is 1,411 times larger than GPT-2's 124 million parameters.

As of now, it is challenging to make predictions about the future of LLMs in the insurance domain. However, based on current trends and recent developments, we can make some assumptions about what the future may hold. LL.M. programs in Europe, for instance, have adapted to generative AI, and students are expected to have knowledge about this technology to succeed in their future careers. Therefore, it is safe to assume that LLM programs may continue to evolve to include more advanced technology and data analysis.

Furthermore, a recent study by MIT researchers developed a system that directly integrates prediction functionality on top of an existing time-series database, which is more accurate and efficient than state-of-the-art deep learning methods when performing two tasks: predicting future values and filling in missing data points (This works especially for medical research). This indicates that the use of machine learning and data analysis may become more prevalent in the legal industry, including the insurance domain.

This project allowed us to learn a little more about LLMs and their functioning. The use of a (certainly limited) LLM on a local machine allowed us to see that the computing power required to run them is not necessarily huge. However, the level of quality of the outputs is not as good as that of the more powerful LLMs (e.g., GPT-4). Nevertheless, by the future outcome, thanks to technological advancements, we can hope to run bigger models on local machines.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008). https://arxiv.org/abs/1706.03762

[2] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[3] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[4] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. https://arxiv.org/abs/2005.14165

[5] Jay Alammar's blog post on "The Illustrated Transformer": http://jalammar.github.io/illustrated-transformer/

[6] Lilian Weng's blog post on "Attention? Attention!": https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

[7] Llama GitHub repository: https://github.com/22-22/llama

[8] Attention is All You Need: https://arxiv.org/abs/1706.03762

[9] Hugging Face, 2023 , Custom Datasets. https://huggingface.co/docs/transformers/custom_datasets.html

[10] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2017). Inferring and Executing Programs for Visual Reasoning. In Advances in Neural Information Processing Systems 30 (NIPS 2017). https://papers.nips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[11] Bender, Emily M., et al. "On the dangers of stochastic parrots: Can language models be too big?." Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. 2021.

[12] gpt-2-simple. (2021). Python package to easily retrain OpenAI's GPT-2 text-generating model on new texts. https://pypi.org/project/gpt-2-simple/

[13] Stack Overflow. (2018). What's ops from tensorflow.python.framework for ? Retrieved from https://stackoverflow.com/questions/50772746/whats-ops-from-tensorflow-python-framework-for

[14] Workiva Inc. Insurance Solutions. Retrieved from https://www.workiva.com/solutions/insurance

[15] Workiva Inc. About Us. Retrieved from https://www.workiva.com/about

[16] ProgrammingHut. (2020, September 25). GPT2 fine tuning | gpt2 text generation | harry potter novel generation gpt2 [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=DNLebQ_vYiw

[17] MIT School of Engineering. (2022, March 22). A tool for predicting the future. MIT News/Adam Zewe. Retrieved from https://computing.mit.edu/news/a-tool-for-predicting-the-future/

[18] Stack Overflow. (2020, February 6). GPT-2 Continue training from checkpoint - python. Retrieved from https://stackoverflow.com/questions/60097717/gpt-2-continue-training-from-checkpoint

[19] Stack Overflow. (2016, November 24). AttributeError: module 'tensorflow' has no attribute 'reset_default_graph'. Retrieved from https://stackoverflow.com/questions/40782271/attributeerror-module-tensorflow-has-no-attribute-reset-default-graph

[20] Workiva, Inc. (2021). 2021 Annual Report. Retrieved, from https://s21.q4cdn.com/997645077/files/doc_financials/2021/ar/2021_10_K_and_AR-(1).pdf

[21] P.Nunes (2022), The Impact of Large Language Models in Insurance https://www.twoimpulse.com/en/insights/large-language-models-insurance

[22] SaaSGenius. (March, 2023). Workiva Reviews. Retrieved, from https://www.saasgenius.com/reviews/workiva/

[23] Deloitte. (2022, May 11). Deloitte Launches New ESG Accelerators for Workiva Platform Users Working To Establish or Enhance Accounting, Financial and Regulatory Reporting. Retrieved from https://www2.deloitte.com/us/en/pages/about-deloitte/articles/press-releases/deloitte-launches-new-esg-accelerators-for-workiva-platform-users-working-to-establish-or-enhance-accounting-financial-and-regulatory-reporting.html

[24] Jk. Rowling. (1997, June 27). Harry Potter and the Sorcerer's Stone [US]. Harry Potter and the Philosopher's stone [UK]. Fantasy book.

[25] The Google Colab : https://colab.research.google.com/drive/12onOmgj9FVcSncGAgJN9b2vv9medYTfj?usp=sharing

[26] The GitHub repository : https://github.com/AverageCoder69/TER_project_ISFA

Annex content

What were the roles of the authors ?

Esther and Mohamad were in charge of the theoretical analysis of large-scale language models (LLMs) to compare them and assess their use in the insurance industry. They began by defining what an LLM is and explaining why it is useful in the field of insurance. They then briefly studied the underlying mathematics of LLMs, referring to "Attention is All You Need" by Vaswani et al. (https://arxiv.org/abs/1706.03762) and the Illustrated Transformer by Jay Alammar (http://jalammar.github.io/illustrated-transformer/) which consist of 10 main steps. Once this was completed, they were ready to compare the GPT-2, Llama, and Bloom models.

Taking into account the various factors and characteristics of each model, they concluded that GPT-2 was the preferred choice for their study. They also examined the Llama model using its GitHub repository as a primary source of information. Finally, they addressed the limitations of LLMs, referring to the paper "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" by Bender et al. (2021), and discussed potential solutions to overcome these challenges.

Jérémy, for his part, was in charge of developing the different models on Google Colab. He arranged the Google Colab to put the code to train the texts. He then analyzed the three texts generated below to try to understand what GPT-2 tried to generate. He also detected errors in the code and tried to correct them and made a report. He also participated in the elaboration of the conclusion with Mohamad and Esther, including an opening sentence on the future of LLMs.