Generative AI Coursera Notes

> [Source: https://www.coursera.org/learn/generative-ai-with-llms/lecture/IrsEw/generative-ai-llms]

1. LLMs and GenAI are general purpose technologies similar to other general purpose technologies like deep learning and electricity.
2. In-Context Learning: How to guide the model to output at inference time with prompt engineering
   and how to tune most important generation parameters of LLMs for tuning your model output.
3. In 2017, the paper "The attention all you need" came.
4. GenAI applications includes chatbots, generating images from text, generate code. So it is capable of creating content that mimics or approximates human ability.
5. Generative AI is a subset of traditional machine learning. LLMs have been trained on trillions of words over many weeks and months and with large amounts of computer power.
   These foundation models include GPT, BERT, LLaMa,BLOOM, FLAN-T5, PaLM etc and
   these foundation models with billions of parameters exhibit emergent properties beyond language alone.
6. Think of Model's parameters as model's memory.
7. While GenAI models are being created for multiple modalities including images, video, audio and speech.
   Here we consider text and natural language generation.
8. LLMs are able to take natural language or human written instructions and perform tasks as much as human would.
9. The text that you pass to an LLM is known as a prompt. The space or memory that is available to the prompt
   is called the context window and this is typically large enough for a few thousand words but differs
   from model to model.
10. The prompt is passed to the model and model predicts the next words and if prompt contains a question then
    model generates an answer. The output of the model is called a completion. The act of using the model to
    generate the text is known as inference. The completion is comprised of the text contained in the original
    prompt, followed by the generated text.

11. LLM use cases and tasks: Essay Writer, text summarization with a given text file, text translation or
    translate text to machine code (code generation), Named-entity recognition, flight information with
    external sources etc.
12. Smaller models with less number of parameters can be fine tuned and then can perform well on specific
    focused task.
13. In many languages, one word can have multiple meanings. These are homonyms.
    Words can have the syntactic ambiguity.
14. The transformer architecture is split into 2 distinct parts: Encoder and decoder.
    Both components share many similarity.
15. First words are converted into numbers using tokenizer which may tell about relative position of words
    in dictionary. This tokenizer also used when we generate text. The input which is represented as numbers
    is passed to embedding layer.
16. Embedding layer is a trainable vector embedding space (a high dimensional space where each token is
    represented as a vector and occupies a unique location within that space). These vectors learn to
    encode the meaning and context of individual tokens in the input sequence.
17. In the original transformer paper, the vector size was actually 512. Using positional encoding, we preserve the information, about the word order and don't lose the position of the word in the sentence.
    Positional encoded vectors are passed to self attention layer and here model analyzes the relationship/contextual dependency between tokens in our input sequence.
18. The self-attention weights are learned during training and stored in these layers that reflect the importance of each word in the input sequence to all other words in the sequence.
19. The transformer architecture has multi-headed self-attention, it means multiple sets of self-attention
    weights or heads are learned in parallel independently of each other. The number of attention heads are
    varies from model to model. But numbers in the range 12-100 are common.
    The intuition here is that each self-attention head will learn a different aspect of language.
    For example, one head may see the people entities in our sentence, another head may focus on the
    activity of the sentence. It's important to note that you don't have to dictate ahead of time what

aspects of language the attention heads will learn. The weights of each head are randomly initialized and
given sufficient training data and time. Each will learn different aspects of the language. While some
attention maps are easy to interpret, others may not be. Now that all of the attention weights have been
applied to your input data, the output is processed through a fully-connected feed-forward network.
The output of this layer is a vector of logits proportional to the probability
score for each and every tokens in the tokenizer dictionary. You can then pass these logits to a final softmax layer, where they are normalized into a probability score for each word.
The output includes a probability for every single word in the vocabulary.

20. Example:
    Machine translation (Sequence to Sequence learning): French to English language - First convert french
    sentence to english tokens, passed to embedding layer and then to multi-head attention layer and then to
    feed forward layer and output of this passed to decoder and decoder predicts next token based on the
    contextual understanding provided by the encoder. For prediction of sequence of tokens, make a loop for decoder for the continuation of decoder process.
21. Transformer Encoder:
    encodes inputs ("prompts") with contextual understanding and produces one vector per input token.
    Decoder: accepts input tokens and generates new tokens and it does in a loop until some stop condition
    is reached.
22. Encoder only models performs sequence-to-sequence modelling with same length of input and output and
    it is useful for classification task like sentiment analysis. BERT is an example of encoder only model
23. Encoder-Decoder models performs sequence-to-sequence tasks such as translation where the input and output
    sequences can be of different lengths. We can scale and train these type of models to perform general text generation tasks. Example: BART as opposed to BERT, T5 etc.
24. Decoder only models: These are most commonly models used today. These models can generalize to most tasks.
    It includes GPT family of models, BLOOM, Jurassic , LLama etc.

25. Prompting and prompt engineering:

The text which we feed into the model is called prompt. The act of generating text is called inference.

The output text is known as completion. The full amount of text or the memory that is available to use

for the prompt is called the context window. Sometimes we don't get the desired output. So we may have

to revise the language in our prompt or the way it is written several times to get the desired output.

This work to develop and improve the prompt is known as prompt engineering. One example to get better

outcome is to include examples of the task that we want the model to carry out inside the prompt.

Providing examples inside the context window is called in-context learning.

Example:

Prompt: Classify this review: I loved this movie!. Sentiment:

Here, Classify this review is an instruction. the context is here a review text: I loved this movie.

And an instruction to produce sentiment at the end is: Sentiment:

This method, including your input data within the prompt is called zero-shot inference.

The largest of the LLMs are surprisingly good at this. Here output of the model is:

Classify this review: I loved this movie!. Sentiment:Positive

Smaller model on the other hand can struggle with this. Here is an example of a completion generated

by GPT-2: Classify this review: I loved this movie!. Sentiment: eived a very nice book review.

Now, to improve performance, we can make longer prompt as:

Classify this review: I loved this movie!. Sentiment:Positive. Classify this review:

I don't like this chair. Sentiment:

Now it has a better chance of understanding task and format of the response we want.

So completion output is:

Classify this review: I loved this movie!. Sentiment:Positive.

Classify this review: I don't like this chair. Sentiment: Negative

Using single example is known as one-shot inference in contrast to the earlier zero shot inference.

Sometimes single example is not well enough for the model to learn what we want.

So, we can include multiple examples. This is known as few-shot inference.

Example:

Classify this review: I loved this movie!. Sentiment: Positive.

Classify this review: I don't like this chair. Sentiment: Negative.

Classify this review: This is not great. Sentiment:
Completion output:
Classify this review: I loved this movie!. Sentiment: Positive.
Classify this review: I don't like this chair. Sentiment: Negative.
Classify this review: This is not great. Sentiment: Negative
In zero shot prompt, context window has few thousands words. In one-shot, few-shot prompts, we fine-tune the model. Fine tuning performs additional training on the model using new data to make it more
capable of the task which we want to perform.

26. As larger and larger models have been trained, it's become clear that the ability of the models to perform multiple tasks
and how well they perform those tasks depends strongly on the scale of the model.
27. Models with more parameters are able to capture more understanding of language. The largest models are

```
surprisingly good at zero shot inference. In contrast, smaller models are
a small number of tasks.

Example: BERT: 110M(110 MILLIONS) and BLOOM: 176B(176 Billions)
```

28. Generative Configuration:

```
If you have used playgrounds for LLMs like hugging face website or an AWS,
with controls like max new tokens, sample top K, sample top P, temperature
These are the configuration parameters that influence the model's output d
These parameters are different than the training parameters that are learn
These configuration parameters are invoked at inference time.
```

- Max new tokens: It can be used to limit the number of tokens that model will generate. You can think of it
as putting a cap on the number of times the model will go through the selection process. The output from the
transformer's softmax layer is a probability distribution across the entire dictionary of the model uses.
Most LLMs by default will operate with so-called greedy decoding. This is the simplest form of next-word
prediction.

greedy: The word/token with the highest probability is selected. This method works very well for short
generation but susceptible to repeated words or repeated sequences of words.
If you want to generate text that's more natural, more creative and avoids repeating words, you need to
use some other controls. Random sampling is the easiest way to introduce some variability.

- Random Sampling: Here model chooses an output word at random using the probability distribution to weight
the selection, So, it selects a token using a random-weighted strategy across the probabilities of all tokens.
Sample top K and Sample top P are sampling techniques that we can use to help limit the random sampling and
increase the chance that the output will be sensible.
- top-K: select an output from the top-k results after applying random-weighted strategy using the probabilities.
if k=3 then model selects highest 3 probabilities tokens. This method can help the model have some randomness
while preventing the selection of highly improbable completion words. This in-turn makes your text generation
more likely to sound reasonable and to make sense. Alternatively, you can use the top-p setting to limit the
random sampling to the prediction whose combined probabilities do no exceed p.
- top-p: select an output using the random-weighted strategy with the top-ranked consecutive results by
probability and with a cumulative probability <= p. For example, if p=0.30 then it selects the words with p=0.20 and p=0.10.
- Temperature: it also controls the randomness of the model output. This parameter influences the shape of the
probability distribution that the model calculates for the next token. Broadly speaking, higher the temperature,
higher the randomness and lower the temperature, lower the randomness. The temperature value is a scaling
factor that is applied within the final softmax layer of the model that impacts the probability distribution
of the next token.
In contrast to the top-p or top-k parameters, changing the temperature actually alters the predictions that the model will make. If we choose a low value of temperature say less than 1 (cooler temperature), then the resulting probability distribution from the softmax layer is more

strongly
peaked with the probability being concentrated in a small number of words.
If we select a higher temperature say >1 then model will calculate a broader flatter probability distribution
for the next token, so it has higher degree of randomness and more variability in the output as compared
to cool temperature setting. It helps to generate text that sounds more creative.
If you leave temperature=1 then it will leave the softmax function as default and unaltered probability
distribution will be used.

29. Generative AI project lifecycle:

- A. Scope:
  Define the use case - LLMs are capable of many tasks but their ability strongly depends on the size
  and architecture of the model. You should think about what the function the LLM will have in your specific
  application like essay writing, summarization, translation, information retrieval like NER, invoke APIs
  and actions etc.
- B. Select:
  Choose an existing model or pre-train your own model - Now your 1st decision will be whether to train
  your own model from scratch or work with an existing base model. In general, most of the time, you will
  start with an existing model but in some cases, it is necessary to train a model from scratch.
  Considerations for choosing a model:

  ```
  A. Foundation Model: Pretrained LLMs
  B. Train your own model from scratch: Custom LLMs
  ```

- C. Adapt and align model:
  Prompt Engineering, Fine-tuning, align with human feedback and Evaluate - Next step is to assess the
  performance of the model. Prompt engineering can be enough sometimes to get your model to perform well.
  So, you'll likely start by trying in-context learning i.e. using examples suited to your task and use case.

Now there are cases where model may not perform well as you need even with one or few shot inference,
in that case, you can try fine-tuning your model. As now model becomes more capable, so align with human
feedback using reinforcement learning.
Now, an important aspect of all these things is Evaluation i.e. some metrics and benchmarks to determine
how well your model is performing.This adapt and align stage is highly iterative.

- D. Application Integration: Optimize and deploy model for inference and Augment model & build LLM-powered
applications.

30. The developers of some of the major frameworks for building GenAI applications like Hugging Face and PyTorch,
have created hubs where you can browse models.
A really useful feature of these hubs is the inclusion of model cards that describe important details
including the best use cases for each model, how it was trained and known limitations. The exact model that
we had chosen will depend on the details of the task you need to carry out.
Variance of the transformer model architecture are suited to different language tasks,
largely because of differences in how the models are trained.
31. Model Architecture and Pre-training (initial training process for LLMs) objectives:
LLM pre-training at high level:
LLM encodes a deep statistical representation of language. This understanding is developed during the models
pre-training phase. When the model learns from vast amounts of unstructured textual data. This can be
Gigabytes, terabytes or petabytes of text. This data is pulled from many sources including scrapes from
the internet and corpora of text that have been assembled specifically for training language models.
In this self-supervised learning step, the model internalizes the patterns and structures present in the
language. These patterns then enable the model to complete its training objective which depends on the
architecture of the model.

During pre-training, model weights get updated to minimize the training loss of the training objective.
The encoder generates an embedding or vector representation for each token. Pre-training also requires a
large amount of compute and the use of GPUs.
Note: When you scrape training data from public sites such as internet, you often need to process data to
increase quality, address bias and remove harmful content. As a result of this data quality curation,
often only 1-3% of tokens are used for pre-training. You should consider this when you estimate how much
data you need to collect if you decide to pre-train to your own model.

32. Models like Encoder only, Encoder-Decoder, Decoder only, all of these models are trained on a different
objective and so learns how to carry out different tasks.

- Encoder-only models, also known as Auto-encoding models and they are pre-trained using masked language modelling(MLM). Here, tokens in the input sequence are randomly masked and the training
objective is to predict the mask tokens in order to reconstruct the original sentence.
This is also called the denoising objective.
Autoencoding models spilled bi-directional representations of the input sequence meaning that the model
has the understanding of the full context of the token and not just the words that come before.
Encoder-only models are ideally suited to task that benefit from this bi-directional contexts.
You can use them to carry out the sentence classification tasks,
for example, sentiment analysis, or token level tasks such as NER(Named Entity Recognition) or word recognition.
Some examples of autoencoder models are: BERT, RoBERTa.
- Decoder-only or auto-regressive models, which are pre-trained using causal language modeling(CLM).
Here, training objective is to predict next token based on the previous sequence of tokens. Predicting the
next token is sometimes called full language modeling by researchers.
Decoder-based auto-regressive models, mask the input sequence and can see the input tokens leading upto the
token in question. The model has no knowledge of the end of the sentence. The model then iterates

over the input sentence one by one to predict the following token. In contrast to the encoder-only architecture, this means that the context is uni-directional.
By learning to predict the next token from a vast number of examples, the model builds up a statistical
representation of the language. Models of this type make use of the decoder component of the original
architecture without the encoder.
Decoder-only models are often useful for text generation. Although larger decoder-only models show
strong zero-shot inference abilities.
Example of these models are: GPT, BLOOM etc.
Final variation of transformer model is sequence to sequence model.

- Sequence to sequence models used both encoders and decoders of the original transformer architecture.
  The exact details of the pre-training objective vary from model to model. A popular seq2seq model T5
  pre-trains the encoder using span corruption which masks random sequences of input tokens.
  Those mask tokens are replaced with a unique sentinel token. Sentinel tokens are special tokens which are
  added to the vocab but do not correspond to any actual word from input text.
  The decoder is then tasked with re-constructing the mask token sequences auto-regressively.
  The output is then the sentinel token followed by the predicted token.
  Sequence-to-Sequence models can be used for translation, text summarization, question-answering etc.
  They are generally useful when we have a body of text as both input and output.
  Example of these models are: T5, BART etc.

33. Larger models of any architecture are typically more capable of carrying out their tasks well.
    Researcher have found that larger a model is, the more likely it is to
    work as needed without additional in-context learning or further training.
34. from 2018 to 2022: BERT-L(340M) --> GPT-2(1.5B) --> GPT-3(175B) --> PaLM(540B)
    And, growth powered by: introduction to transformer, assess to massive datasets, more powerful compute
    resources. Training of enormous model is difficult and very expensive.
35. Computational Challenges of training LLMs:
    When we try to train large language models then it is running out of memory, we get,
    OutOfMemoryError: CUDA out of memory. If you have ever tried training or even just
    loading your model on nvidia gpus, this error message might look familiar.

CUDA stands for compute unified device architecture, is a collection of libraries and tools developed by
Nvidia GPUs. Libraries such as PyTorch,Tensorflow use CUDA to boost performance on matrix multiplication
and other operations common to deep learning. Most LLMs are huge and require a ton of memory to store
and train all of their parameters.

36. Approximate GPU RAM needed to store 1B parameters:

```
1  parameter = 4 bytes (32-bit float)
1B parameters = 4*10^9 bytes = 4 GB (This is a lot of memory)

Now, we only accounted for the memory to store the model weights so far. I
we'll have to plan for the additional parameters that use GPU memory durin
2 Adam optimizer states (+8 bytes per parameter), Gradients (+4 bytes per
temp memory (variable size): (+8 bytes per parameter (high-end estimate)).
It leads to 20 extra bytes of memory per model parameter.

So, total we need 24 GB to train a model on 32-bit precision system.
```

37. So, what options do you have to reduce the memory required for training:
    A. Quantization: reduce 32-bit floating point precision number to 16-bit floating point precision number
    or 8-bits integer numbers.
    The corresponding data types used in deep learning libraries are FP32 for 32 bit full floating point precisions, FP16 or BFLOAT16 for 16 bits floating point half precision and INT8 for 8-bit integers.
    By default model weights, activations and other model parameters are stored in FP32.
    Recently, BFLOAT16 becomes alternative to FP16. BFLOAT16 stands for Brain Floating point precision
    developed at google brain has become a popular choice in deep learning.
    Many LLMs including FLAN-T5 have been pre-trained with BFLOAT16 (BF16). BF16 is a hybrid between
    half precision FP16 and full precision FP32. BF16 significantly helps with training stability and
    it is supported by newer GPUs like NVIDIA A100. BFLOAT16 is often described as a truncated 32-bit float
    as it captures the full dynamic range of the full 32-bit float that uses only 16 bits. BFLOAT16 uses the
    full 8-bits to represent exponent but truncates the fraction to just 7 bits.

It saves memory and increases the model performance by speeding up calculations.
The downside is BF16 is not well suited for integer calculations but it is rare in deep learning.
Quantization-aware training (QAT) learns the quantization scaling factors during training.

38. As model sizes get larger, you will need to split your model across multiple GPUs for training.
    It could require 100s of GPUs which is very expensive.
    (For 175B param model, 4200 GB memory at 32 bits system) and for 500B param model, 12000 GB memory
    is needed at 32 bits system.
39. Even if your model does fit onto a single GPU, there are benefits to using multiple GPUs to speed up
    your training.
    Distributed Data Parallel (DDP):
    Let's begin with the case when your model is still fits on a single GPU. The first step in scaling model
    training is: Distribute large datasets across multiple GPUs and process these batches of data in parallel.
    A popular implementation of this model replication technique is: PyTorch's distributed data parallel (DDP).
    DDP copies your data onto each GPU using dataloader and sends baches of data to each GPU in parallel.
    Each dataset is processed (forward and backprop) in parallel and then synchronization step combines the
    result of each GPU (synchronizing gradients) which in turn updates the model on each GPUs which is always
    identical across chips. This implementation allows parallel computation across all GPUs that results in
    faster training.
40. If model does not fit in a single GPU then you should look into another technique called model sharding.
    A popular implementation of sharding is Pytorch's Fully Sharded Data Parallel(FSDP).
    It is motivated by the paper by microsoft in 2019 and it has used the
    technique ZeRO -- zero data overlap between GPUs.
    ZeRO stands for zero redundancy optimizer and the goal of ZeRO is to optimize memory by distributing
    or sharding model states across GPUs with zero data overlap.
    One limitation of the model replication strategy is that we need to keep a full model copy on each GPU which
    leads to redundant memory consumption. ZeRO eliminates this redundancy by

distributing(sharding) the model
parameters, gradients, optimizer states across GPUs instead of replicating them. FSDP follows this technique.
Each GPU requests data from the other GPUs on-demand to materialize the sharded data into unsharded data
for the duration of the operation. After the operation, you release the unsharded non-local data back to
the other GPUs as the original sharded data. You can also choose it to keep it for future operations during
backward pass. This requires more GPU RAM again. This is a typical performance vs memory trade-off decision.
In the final step, FSDP synchronizes the gradients across the GPUs in same way as DDP.
FSDP helps to reduce overall GPU memory utilization. FSDP supports offloading to CPU if needed. To manage the
trade off between performance and memory utilization, you can configure level of sharding via sharding factor.
sharding factor=1 removes the sharding and replicates the full model similar to DDP.
If sharding factor = max number of available GPUs, you turn on full sharding. This has the most memory savings
but increases the communication volume between GPUs.
when model size (no. of parameters) is high then teraflops/gpu value is high for full sharding.
1 treaflops = 1 trillion floating point operations.

41. Scaling laws and compute-optimal models:

```
The goal during pre-training is to maximize the model's performance of its
minimizing the loss when predicting tokens.

Two options you have to achieve better performance are increasing the size
trined the model and increasing the parameters in your model. In theory, y
these quantities to improve performance. However, another issue to take in
compute budget which includes factors like the number of GPUs you have acc
available for training models.
```

42. Compute budget for training LLMs:
    A. petaflop/s-day ==> It is a measurement of number of floating point operations performed at the rate of
    1 petaFLOP per second for one day.
    1 petaFLOP/s = 1 quadrillion floating point operations per second = It is approximately equivalent to

8 NVIDIA V100 GPUs operating at full efficiency for one full day. If we have more powerful processor then it can carry out more operations at once.
For example 2 NVIDIA A100 GPUs give equivalent compute to the 8 V100 chips
Bigger models takes more computing resources to train and generally also require more data to achieve good
performance. Researchers have found that many of the hundred billion parameters LLMs like GPT-3 may actually
be over parameterized meaning they have more parameters than they need to achieve a good understanding
of language and under trained so that they would benefit from seeing more training data. The authors of the
paper hypothesized that smaller models may be able to achieve the same performance as much larger ones if
they are trained on larger datasets.

43. Pre-training for domain adaptation:
Here, we may find necessary to pre-train our own model from scratch. If your target domain uses vocabulary
and language structures that are not commonly used in day to day language. We may need to perform domain
adaptation for good performance. For example, imagine you are developer and building an app to help
lawyers to summarize legal briefs. Legal writing makes use of very specific terms like "mens rea" and
"res judicata". These words are rarely used outside of the legal world means they are unlikely to have
appeared widely in the training text of existing LLMs. As a result models may have difficulty understanding
these terms or using them correctly. Another issue is that legal language sometimes uses everyday words
in a different context.
For similar reasons, you may face the challenges if you try to use an existing LLM in a medical application.
Medical language contains many uncommon words to describe medical conditions and procedures.
And these may not appear frequently in training datasets consists of web scrapes and textbooks.
Example: Doctor generally writes: 1 tab po qid pc & hs... medical store people understands it as "take one tablet by mouth four times a day, after meals and at bedtime".

Because models learn their vocabulary and understanding of language through the original pre-training task.
Pre-training your model from scratch will result in better models for highly specialized domains like law, medicine, finance or science etc.

44. BloombergGPT: domain adaptation for finance:
    BloombergGPT was first announced in 2023 in a paper by colleagues at bloomberg, new york.
    BloombergGPT is an example of LLM that has been pre-trained for a specific
    domain, in this case, finance. Researchers have chosen 51% financial data (public+private) and 49% public data.
    BloombergGPT, developed by Bloomberg, is a large Decoder-only language model.