**How Transformer LLMs Work**

Introduction

The Transformer Architecture was first introduced in the 2017 paper, Attention is All You Need by Ashish Vaswani et al for machine translation tasks.

The idea it to input an English sentence and have the network output a German sentence.

This same architecture tend to be great at inputting a prompt and outputting a response to that prompt, like a question and the answer to that question.

The original transformer consists of an encoder and a decoder. The encoder preprocesses the entire input English text to extract the context needed to perform the translation. Then the decoder uses the encoder context to generate the German. The encoder + decoder form the basis for the models used in many language models today.

The Encoder model provide rich, context sensitive representations of the input text, and is the basis for the BERT model and most of the embedding models using RAG applications.

The Decoder model performs text generation tasks such as summarizing text, writing code, answering questions, and is the basis for most popular LLMs, such as those from OpenAI, Anthropic.

A diagram of a computer model

AI-generated content may be incorrect.

Course outline:

* Delve into recent developments in LLMs
* Learn about tokenization
* Gain intuition about how transformers work, focusing on the decoder only models.
  + A generative takes in the text prompt, and it generates a text in response by generating one token at a time.
  + How generation process works: The model starts by mapping each input token into an embedding vector that captures the meaning of that token.
  + After that, the model parses these token embeddings through a stack of transformer blocks, where each block is a specific neural network architecture that those designed to learn flexibly from data and also scale well on GPUs.
  + Each transformer block is made up of an attention layer and a feed-forward network. The model then uses the output vectors of the transformer blocks and passes them to the last component, the language modelling head.
  + The language modelling head generates the output token

A diagram of a transformer

AI-generated content may be incorrect.

**Lesson 1: Understanding Language Models: Language as a Bag-of-Words**

Learn about the evolution of how language has been represented numerically.

**Bag-of-words:** An algorithm that represents words as large sparse vectors or arrays of numbers, which simply record the presence of words.

**Word2Vec:** Word representation capture the meaning of words in the context of a few neighbouring words.

**Transformers:** Dense vectors captured the meaning of words in the context of a sentence or a paragraph.

A diagram of a language

AI-generated content may be incorrect.

Language is a tricky concept for computers. Text is unstructured in nature and loses its meaning when represented by zeros and ones or individual characters. As a result, throughout the history of language AI, there has been a large focus on representing language in a structured manner so that it can be more easily used by computers.

From generating text, to creating numerical representations, and classifying textual inputs. These are just a few of the numerous tasks you can do with language AI.

At the start of this language AI field, the focus was mainly on representing language to analyse unstructured data. A first and still very relevant method is by representing language as a bag-of-words.

A diagram of a language

AI-generated content may be incorrect.

**Tokenization:** The process of converting the input text into pieces. Each individual word is called a token. A token can be any even smaller than an entire word.

You can perform the same tokenization process with another document. Now with 2 sets of tokens, you can create something called a vocabulary. Vocabulary contains all unique words found in both input documents. As such, vocabulary would be lesser than the amount of tokens generated, which is referred to as the vocabulary size.

A diagram of a cat

AI-generated content may be incorrect.

“My cat is cute” -> This input has 4 tokens that match with some of the words in the vocabulary. You can then count how many times of a certain token appears in the vocabulary that we already created.

A screenshot of a computer

AI-generated content may be incorrect.

However, you would also need to take note of the vocabulary that do not appear in the input. A sentence not only gives meaning to the words it contains, but also the words it doesn’t.

Bag-of-words: A numerical representation that indicates the count of individual words appearing in the vocabulary. It does nothing more other than this.

The order is important as it allows us to compare different sentences to one another.

In practice, we call this a vector representation -> A list of numerical values that represents the input.

**Lesson 2: Understanding Language Models: (Word) Embeddings**

Bag-of-words has a flaw: It does not consider the semantic nature of text. It considers language to be nothing more than an almost literal bag-of-words, and ignores the semantic nature or meaning of text.

Word2Vec is one of the first successful attempts at capturing the meaning of text in vector embeddings through neural networks.

* To do so, Word2Vec learned semantic representations of words by training on vast amounts of textual data (like the entirety of Wikipedia).
* To generate these semantic representations, Word2Vec leverages neural networks.
* Neural networks: Networks consist of interconnected layers of nodes that process information. Neural networks can have many layers, which each connection has a certain weight depending on the inputs. The weights are often referred to as parameters of the model.

A diagram of a machine

AI-generated content may be incorrect.

* Using these neural networks, Word2Vec generates word embeddings by looking at which other words they tend to appear next to in a given sentence. You start by assigning every word in your vocabulary with a vector embedding.
* Example, 5 values for each word is initialized with random values. Then in every training step, you take pairs of words from training data, and the model attempts to predict whether or not they are likely to be neighbours in a sentence.
* During this training process, Word2Vec learns the relationship between words and distils that information into the embedding.
* If the 2 words tend to have the same neighbour, their embeddings will be closer to one another and vice versa.

A close-up of a sign

AI-generated content may be incorrect.

The resulting embeddings capture the meaning of words.

Example

Assume that you have an embedding for the word “cats”. This embedding has generates values between -1 and 1.