HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Large Language Models and Generative AI for NLP

Aarne Talman

Machine Learning & Data Science Lead, EMEA Centre for Advanced AI at Accenture
Visiting Researcher and Lecturer at University of Helsinki

Contents

1. Course Practicalities
2. Introduction to Large Language Models and Generative AI
3. Lab

Contents

1. Course Practicalities
2. Introduction to Large Language Models and Generative AI
3. Lab

Teachers

Aarne Talman, Accenture

Responsible for weeks 1, 2, 4 & 7

Dmitry Kan, TomTom

Responsible for weeks 5 & 6

Jussi Karlgren, Silo AI

Responsible for week 3

Aarne Talman

- PhD in Language Technology
- Machine Learning Lead at Accenture EMEA Centre for Advanced AI
- 20 years of industry experience
- Formerly:
  - Head or Tech at SiloGen (Silo AI)
  - UK CTO at Nordcloud
  - Strategy consultant at Gartner and Accenture
  - Search Product Manager at Nokia

Dmitry Kan

- PhD in Applied Mathematics (2011, machine translation)
- Senior Product Manager, Search & ML, TomTom
- Host of Vector Podcast: https://www.vectorpodcast.com/
- Blogging: https://dmitry-kan.medium.com/
- 20 years of experience in developing search engines and software for start-ups and multinational technology giants
- Previous roles:
    - Principal AI Scientist / Head of Engineering, Silo.AI
    - Head of Search and AI, AlphaSense

Jussi Karlgren

- PhD in Computational Linguistics
- Docent at University of Helsinki
- Principal AI Scientist at Silo AI
- Based in Stockholm
- Over 30 years of Industry experience
- Formerly at Spotify, Yahoo, Xerox PARC, IBM, Stanford ...
- Most interested in
  - linguistic aspects of stylistic and situational variation
  - evaluation of information systems
- Both will come up in week 3!

Course description

This **hands-on** course explores Large Language Models (LLMs) and their applications in Natural Language Processing (NLP)

You will learn:

- What are LLMs and what is Generative AI?
- How LLMs work?
- How to fine-tune them for specific tasks?
- How to use them for various NLP applications?

Syllabus

Week 1: Introduction to Generative AI and Large Language Models (LLM)

- Introduction to Large Language Models (LLMs) and their architecture
- Overview of Generative AI and its applications in NLP
- Lab: Tokenizers

Week 2: Using LLMs and Prompting-based approaches

- Understanding prompt engineering and its importance in working with LLMs
- Exploring different prompting techniques for various NLP tasks
- Hands-on lab: Experimenting with different prompts and evaluating their effectiveness

Week 3: Evaluating LLMs

- Understanding the challenges and metrics involved in evaluating LLMs
- Exploring different evaluation frameworks and benchmarks
- Hands-on lab: Evaluating LLMs using different metrics and benchmarks

Week 4: Fine-tuning LLMs

- Understanding the concept of fine-tuning and its benefits
- Exploring different fine-tuning techniques and strategies
- Hands-on lab: Fine-tuning an LLM for a specific NLP task

Week 5: Retrieval Augmented Generation (RAG)

- Understanding the concept of RAG and its advantages
- Exploring different RAG architectures and techniques
- Hands-on lab: Implementing a RAG system for a specific NLP task

Week 6: Use cases and applications of LLMs

- Exploring various real-world applications of LLMs in NLP
- Discussing the potential impact of LLMs on different industries
- Hands-on lab: TBD

Week 7: Final report preparation

- Students work on their final reports, showcasing their understanding of the labs and the concepts learned.

Prerequisites

- Python coding experience
- Basics of machine Learning
  - e.g. Machine Learning for Linguists (LDA-T317, KIK-LG210)

Weekly Schedule

- Lectures:
  - Time: Tuesday at 2:15 PM – 3:45 PM
  - Location: Metsätalo, B308 (sali 12), Unioninkatu 40 (Metsätalo)

- Labs:
  - Time: Thursday at 2:15 PM – 3:45 PM
  - Location: Kielikeskus, sh.405, Language Centre, Fabianinkatu 26

Course Evaluation

- You will need to submit **a final report** that covers all the labs:
  - What was done in each lab?
  - What was the motivation behind your solutions?
  - What did you learn?
  - Reflect on the challenges you encountered?

- You will also be required to submit **a link to your code in Github** that covers all the labs
  - You will need to fork the course Github repo and modify the code as required for the lab exercise: https://github.com/Helsinki-NLP/LLM-course-2024/

- Final report submission deadline
  - **31st December 2024**

Materials

- Slides and Code:
  - All presentation material and code will be made available in course GitHub: https://github.com/Helsinki-NLP/LLM-course-2024/

Contents

1. Course Practicalities
2. Introduction to Large Language Models and Generative AI
3. Lab

Outline

1. What are language models?
2. Popular language modeling algorithms
3. The Transformer Architecture
    - Core Components of the Transformer Architecture
    - Variants of the Transformer Architecture
    - Decoder Only LLMs (“Generative models”)
4. Recent developments
    - Mixture of Experts (MoE) Models
    - Multimodal models
5. Lab assignment / Homework
    - Tokenizers
6. References for further study

What are language models?

- **Language modeling** is the task of **estimating the probability distribution of sequences of words or tokens in a given language**.
- More formally, given a sequence of words $\mathbf{w_1 w_2 \ldots w_n}$ in a language ***L***, a language model assigns a probability $\mathbf{P(w_1 w_2 \ldots w_n)}$ to the entire sequence.
- This can also be expressed as the **conditional probability of the next word given the previous words**:

$$\mathbf{P(w_n \mid w_1 w_2 \ldots w_{n-1})}$$

- In essence, **language modeling aims to capture the statistical patterns and structures of a language**, allowing it **to predict the likelihood of a given sequence of words** or to **generate new text** based on existing sequences in the language.

Popular language modeling algorithms (1/3)

N-gram models

- Core Idea: Predict the next word based on the frequencies of sequences of n previous words (n-grams) in the training corpus.

Hidden Markov Models (HMMs)

- Core Idea: Model language as a sequence of hidden states with probabilistic transitions between them. Each state emits words with certain probabilities.

(Feed-forward) Neural Language Models (NLMs)

- Core Idea: Use feed-forward neural networks to predict the probability distribution of the next word given the previous context.
- Key Reference:
  - Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model.

Example n-grams: 2-gram (bigram)

The quick *brown fox jumps over the lazy dog*

The ***quick brown*** *fox jumps over the lazy dog*

The quick ***brown fox*** *jumps over the lazy dog*

The quick brown ***fox jumps*** *over the lazy dog*

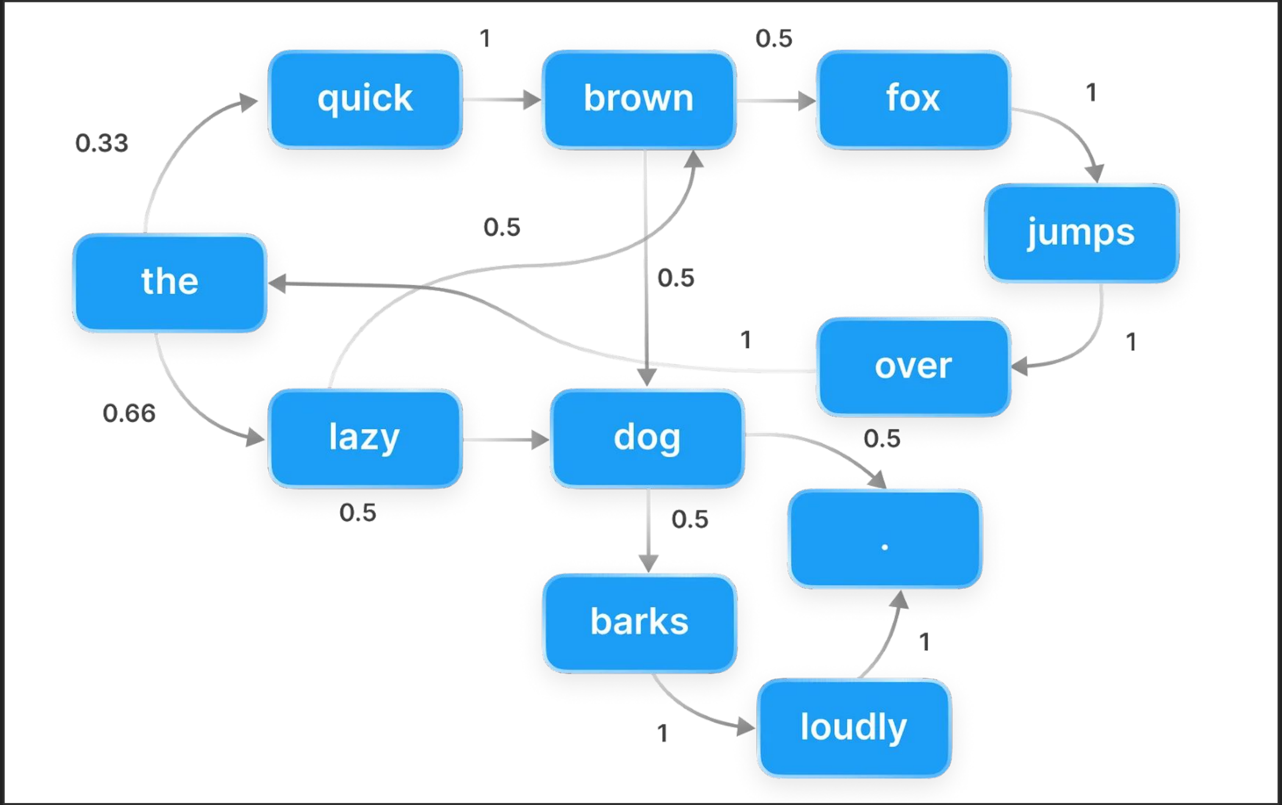
The quick brown fox ***jumps over*** *the lazy dog*

The quick brown fox jumps ***over the*** *lazy dog*

The quick brown fox jumps over ***the lazy*** *dog*

The quick brown fox jumps over the ***lazy dog***

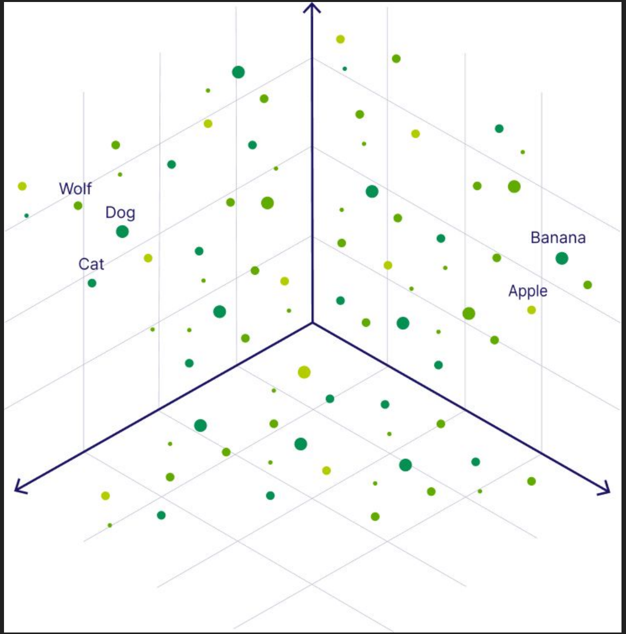
Example Markov Chain



Popular language modeling algorithms (2/3)

Word Embeddings (e.g., Word2Vec, GloVe)

- Core Idea: Represent words as dense vectors in a continuous space, capturing semantic relationships between words.
- Key References:
  - Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.
  - Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation.



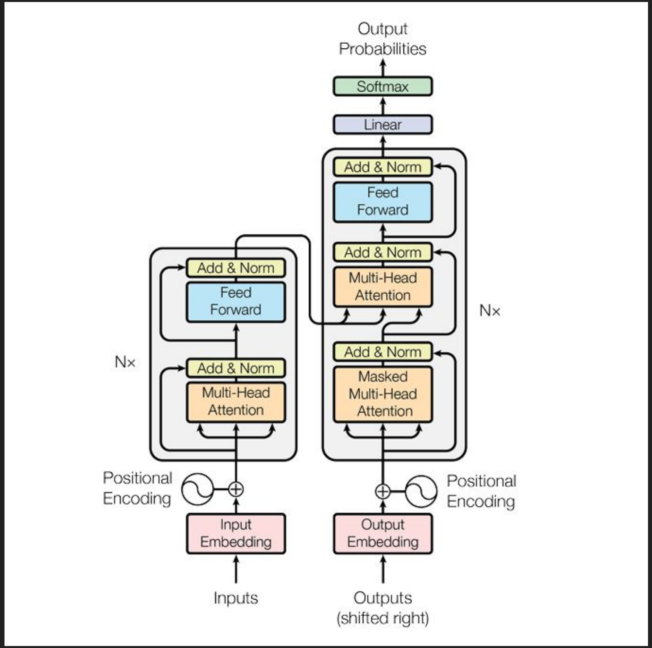
Recurrent Neural Networks (RNNs) with Language Modeling objective

- Core Idea: Use sequential processing to capture dependencies between words in a sentence, allowing for variable-length input and potentially capturing long-range context.
- Key Reference:
  - Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory

Popular language modeling algorithms (3/3)

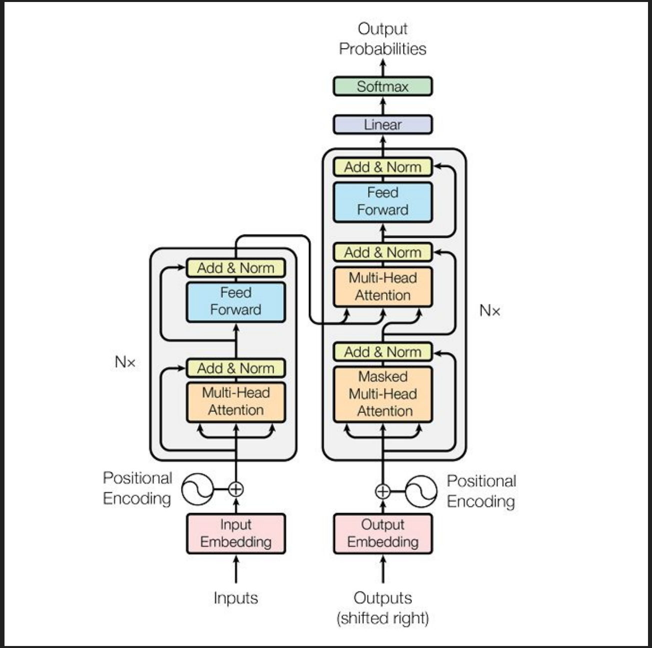
Transformer models

- Core Idea: Utilize self-attention mechanisms to weigh the importance of different words in a sentence, enabling the capture of long-range dependencies and contextual relationships effectively.
- Originally designed to be a machine translation model
- Key Reference:
  - Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need



Transformer Architecture

- **Captures long-range dependencies:** The self-attention mechanism allows the model to effectively capture relationships between words that are far apart in the sequence, overcoming limitations of previous models like RNNs.
- **Parallel processing:** The Transformer architecture is highly parallelizable, enabling faster training and inference on modern hardware.
- **Strong performance on various NLP tasks:** Transformers have achieved state-of-the-art results on a wide range of NLP tasks, including machine translation, text summarization, question answering, and many others.



Source: Vaswani et al. 2017

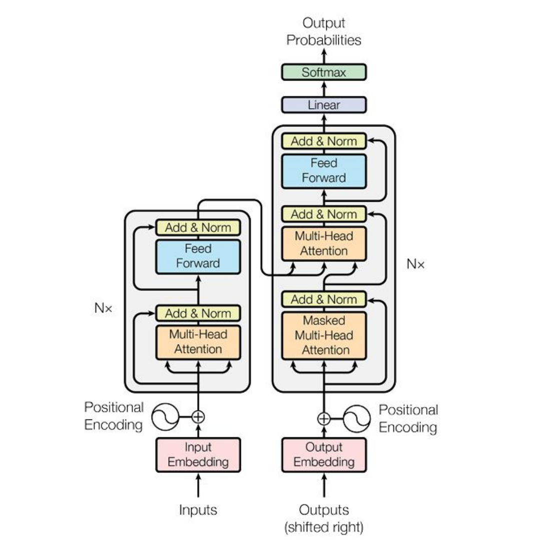
Core Components of the Transformer Architecture (1/3)

Self-Attention Mechanism:

- The heart of the Transformer, enabling it to **weigh the importance of different words in a sequence** when making predictions.
- **Each word in the input sequence attends to every other word**, calculating attention scores that represent the relative importance of each word for understanding the current word.
- This allows the model to **capture long-range dependencies** and contextual relationships effectively.
- Further study: https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

Multi-Head Attention:

- Employs **multiple attention heads in parallel**, each focusing on different aspects of the input sequence.
- This allows the model to learn a richer and more nuanced representation of the input.



Source: Vaswani et al. 2017

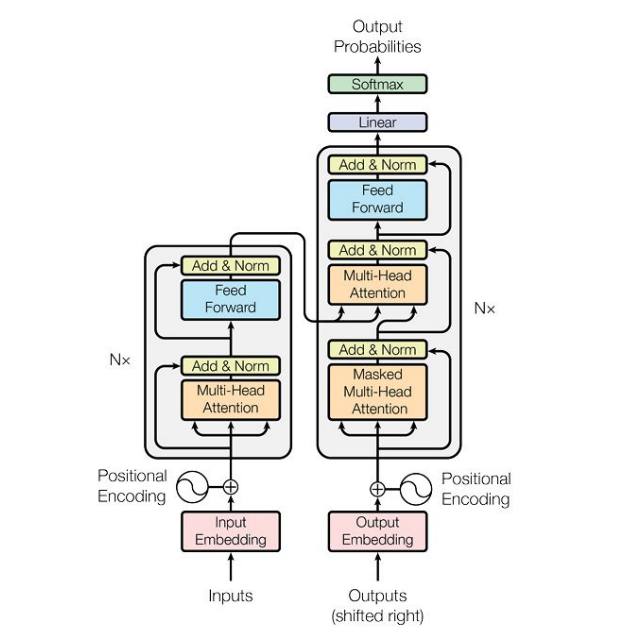
Core Components of the Transformer Architecture (2/3)

Positional Encoding:

- Since **self-attention is permutation-invariant**, it needs a way to incorporate word order information.
- Positional encodings are added to the input embeddings, providing the model with **information about the position of each word in the sequence**.

Encoder and Decoder:

- The Transformer typically consists of **an encode**r and a **decoder stack**, each **composed of multiple layers**.
- The e**ncoder processes the input sequence**, generating a contextualized representation for each word.
- The **decoder generates the output sequence**, attending to both the encoder output and its own previous outputs.



Source: Vaswani et al. 2017

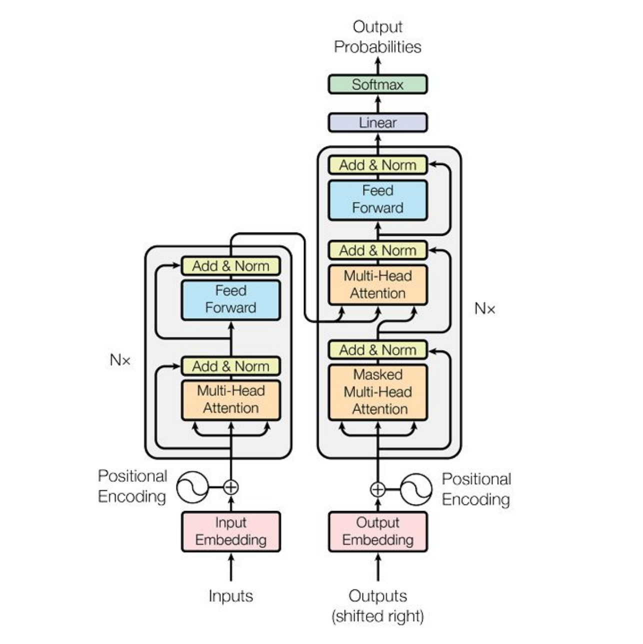
Core Components of the Transformer Architecture (3/3)

Feed-Forward Networks:

- Each layer in the encoder and decoder includes a **feed-forward network that applies non-linear transformations to the input**, further enhancing the model's representational power.

Layer Normalization and Residual Connections:

- **Layer normalization helps stabilize** training and improve the model's robustness.
- **Residual connections prevent vanishing gradients** by bypassing components of the network and allowing gradients to flow more easily during training, enabling the training of deeper networks.

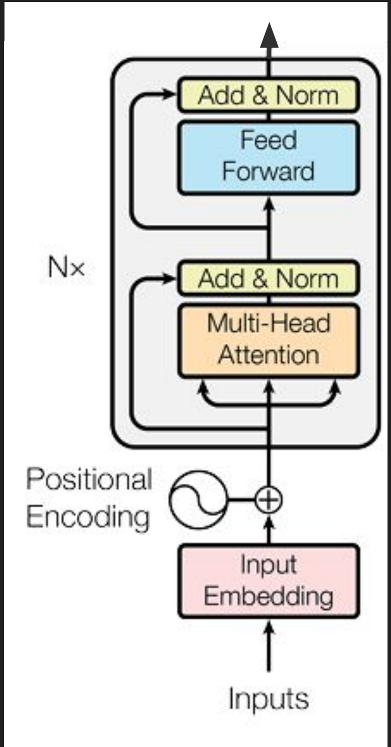


Source: Vaswani et al. 2017

Variants of the Transformer Architecture

Encoder-only Transformers ("Embedding models"):

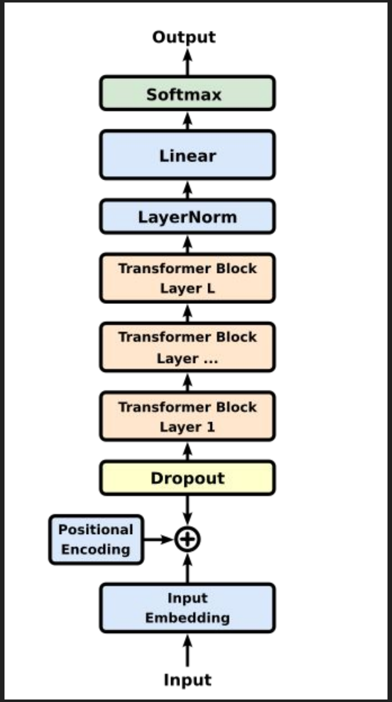
- Structure: **Only utilize the encoder stack** of the original Transformer architecture.
- Focus: Primarily designed **for** understanding and **representing the input text, capturing contextual relationships between words**.
- Common Applications:
  - Text classification: E.g. determining the sentiment or topic of a given text.
  - Named entity recognition: Identifying and classifying entities in text (e.g., names of people, organizations, locations).
- Examples: BERT, RoBERTa



Variants of the Transformer Architecture

Decoder-only Transformers ("Generative models"):

- Structure: Only utilize the decoder stack of the original Transformer architecture.
- Focus: Specialize in generating text, predicting one word at a time based on the previous context.
- Common Applications:
  - Text generation: Generating creative writing, completing sentences, or writing code.
  - Text summarization: Condensing a longer text into a shorter, informative summary.
- Examples: GPT family (GPT, GPT-2, GPT-3), LLaMa family

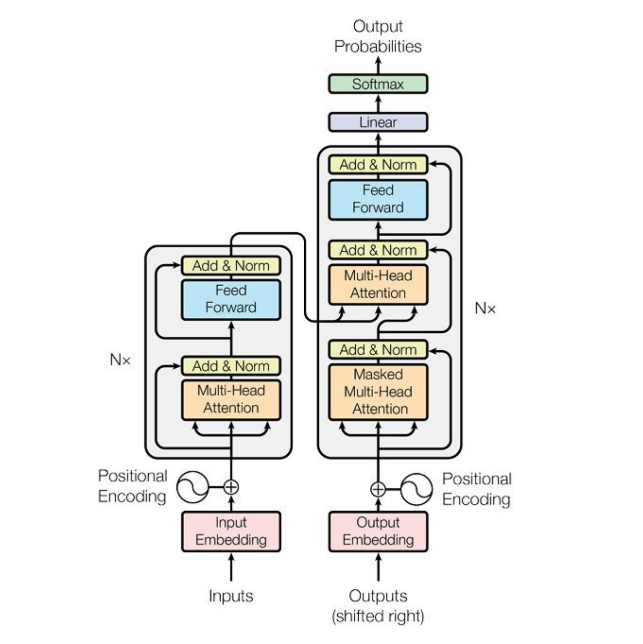

Output
Softmax
Linear
LayerNorm
Transformer Block Layer L
Transformer Block Layer ...
Transformer Block Layer 1
Dropout
Positional Encoding
Input Embedding
Input

GPT-1

Variants of the Transformer Architecture

Encoder-Decoder Transformers:

- Structure: Utilize both the encoder and decoder stacks of the original Transformer.
- Focus: Combine the capabilities of understanding the input text (encoder) and generating output text (decoder).
- Common Applications:
  - **Machine translation:** More powerful for translating between languages with different structures.
  - **Dialogue systems:** Generating responses in a conversation based on the context of the dialogue.
  - **Summarization:** Generating summaries that are both informative and abstractive.
- Examples: T5, BART, Neural Machine Translation (NMT) models



Source: Vaswani et al. 2017

Decoder Only LLMs ("Generative LLMs")

GPT (Generative Pre-trained Transformer) (2018)

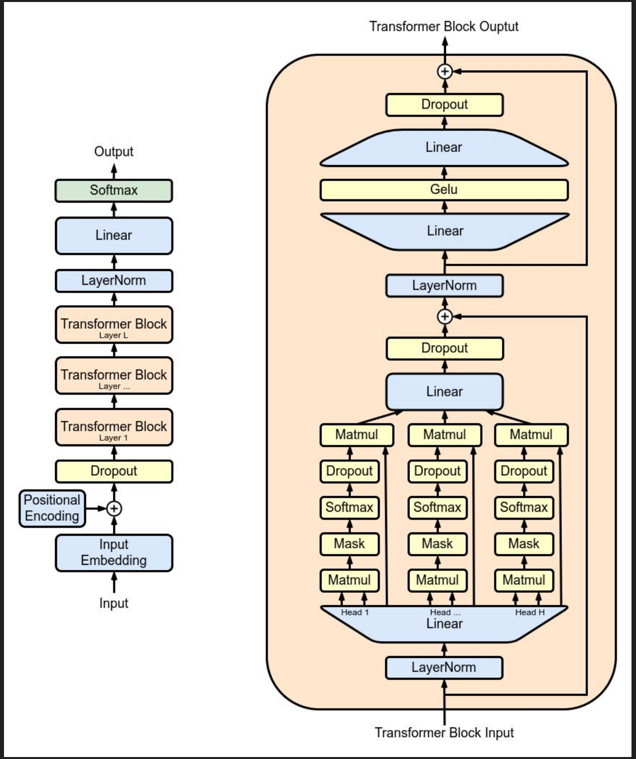
- Standard decoder-only Transformer with masked self-attention.
- Pre-trained on a large corpus of text data using the language modeling objective.
- Fine-tuned on specific tasks using supervised learning.

GPT-2 (2019)

- Scaled-up version of GPT with more layers and parameters.
- Demonstrated the power of larger models for language generation.

GPT-3 (2020)

- Massively scaled-up version of GPT with 175 billion parameters.
- Showcased the potential of few-shot learning, where the model can perform new tasks with just a few examples.



Decoder Only LLMs (“Generative LLMs”)

LLaMA (Large Language Model Meta AI) (2023)

- Focuses on efficiency and performance, achieving comparable results to much larger models with fewer parameters.
- Employs various optimizations and architectural choices to improve training and inference efficiency.

GPT-4 (2023)

- Details of the architecture are not fully disclosed, but it is expected to be a further scaled-up and improved version of the GPT-3 architecture.
- Demonstrates advancements in multimodality, handling both text and image inputs.

Mistral (2023)

- Mistral aims to achieve the best possible performance with the fewest parameters, making it more accessible and less computationally expensive to run than larger models.
- Open-weight and open source. Many of Mistral's models are open-weight, meaning the weights are publicly available, and some are even open source.

Recent developments: Mixture of Experts (MoE) Models

- Core Idea: A type of model architecture where **multiple "expert" networks specialize in different subtasks** or domains. A "gating" network dynamically routes inputs to the most relevant experts for processing.
- Key Differences from Standard Decoder-Only Models:
    - **Sparsity:** MoE models activate only a subset of experts for each input token, leading to more efficient computation and parameter usage compared to dense models that activate all parameters for every token.
    - **Specialization:** Experts can specialize in different linguistic patterns, topics, or styles, enabling the model to handle a wider range of tasks and generate more diverse and nuanced outputs.
    - **Adaptability: T**he gating network learns to dynamically route inputs based on their content, allowing the model to adapt to different contexts and tasks.
- Examples:
    - **GPT-4**: While not a fully MoE model, GPT-4 reportedly incorporates a Mixture of Experts layer in its architecture, potentially contributing to its improved capabilities.
    - **Mixtral**: A sparse mixture-of-experts model developed by Mistral AI, aiming for efficiency and strong performance across various tasks.

Recent developments: Multimodal models

- **Core Idea:** A type of language model that **can process and integrate information from multiple modalities**, such as text, images, audio, and video.
- Key Differences from Standard Decoder-Only Models:
  - **Multimodal Input Processing:** These models incorporate specialized encoders for each modality (e.g., vision encoders for images, audio encoders for sound) to extract meaningful representations. These representations are then fused and integrated with textual information.
  - **Cross-Modal Understanding:** Multimodal LLMs can reason across different modalities, for example, by answering questions about images, generating image captions, or even translating between modalities.
  - **Enhanced Contextual Awareness:** By combining information from multiple sources, these models gain a richer understanding of context, leading to more accurate and nuanced responses.
- Examples:
  - **GPT-4V:** An extension of GPT-4 with vision capabilities, allowing it to analyze images, answer questions about visual content, and generate detailed descriptions.
  - **Flamingo:** A visual language model from DeepMind that can connect information from images and text to perform tasks like visual question answering and image captioning.

Contents

1. Course Practicalities
2. Introduction to Large Language Models and Generative AI
3. Lab

Positional Encodings

- Unlike Recurrent Neural Networks (RNNs), the Transformer architecture processes **all input tokens simultaneously**. This loses the inherent sequential information of the input.
- Positional encodings are added to the input embeddings to provide information about the position of each token in the sequence.
- This is crucial for tasks where word order is critical, such as:
  - Natural Language Understanding: Parsing grammatical structure, resolving coreferences.
  - Machine Translation: Preserving the correct word order in the target language.
  - Text Summarization: Identifying the key points in a text based on their position and relationship to other elements.
- **How it works:** Each position in the input sequence is mapped to a unique vector of values. These vectors are then added (summed) to the corresponding word embeddings (they have the same dimensionality).

Positional Encodings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- Formula Breakdown:
  - **PE(pos, 2i)** and **PE(pos, 2i+1)**: These represent the values of the positional encoding vector for a given token at position pos. The 2i and 2i+1 indicate even and odd dimensions of the encoding vector, respectively.
  - **pos**: This is the position of the token in the input sequence (e.g., 0 for the first token, 1 for the second, and so on).
  - **i**: This represents the dimension of the positional encoding vector. It ranges from 0 to d_model/2, where d_model is the dimensionality of the word embeddings (typically 512 or 768).
  - **d_model**: This is the dimensionality of the word embeddings and also the positional encoding vectors. This ensures that they can be added together.
  - **10000**: This is a constant scaling factor used in the formula.

Positional Encodings

- **Sinusoidal Functions:**
  - The formula uses sine and cosine functions to generate a unique pattern of values for each position in the sequence. The use of different frequencies (determined by 2i/d_model) allows the model to capture both absolute and relative positional information.
- **Even and Odd Dimensions:**
  - The even dimensions of the positional encoding vector use the sine function, while the odd dimensions use the cosine function. This creates a wave-like pattern that provides a rich representation of the position.
- **Adding to Word Embeddings:**
  - These generated positional encoding vectors are then added to the corresponding word embeddings. This combined representation is fed into the Transformer's encoder and decoder layers.

Positional Encodings

- **Unique Encoding:** Different positions get distinct encoding vectors due to the periodic nature of sine and cosine functions.
- **Distance Awareness:** The model can learn relative positions between tokens by attending to the differences in their positional encodings.
- **Extrapolation:** The sinusoidal functions allow the model to generalize to sequences longer than those seen during training.

Why are Transformers more parallelizable than RNNs?

- **RNNs and Sequential Processing:**
  - Sequential Nature: **RNNs process information step-by-step**. Each hidden state depends on the previous one, creating a chain-like structure.
  - Limited Parallelization: While you can parallelize computations within a single time step, you cannot process multiple time steps concurrently.
- **Transformers and Parallel Processing:**
  - Transformers rely on the self-attention mechanism, which allows the model to **weigh the importance of different parts of the input sequence simultaneously**. This **eliminates the need for sequential processing**.
  - Unlike RNNs, transformers **do not have recurrent connections**. This means that the computation for each position in the sequence can be performed independently and in parallel.
  - The lack of sequential dependencies makes transformers highly efficient for processing long sequences, as the entire input can be processed in parallel.

Lab Assignment / Homework

- Research on Tokenizers and write a section to your final report reflecting on the following questions:
  - What are tokenizers?
  - Why are they important for language modeling and LLMs?
  - What different tokenization algorithms there are and which ones are the most popular ones and why?

- Some references:
  - Neural Machine Translation of Rare Words with Subword Units:
    - https://arxiv.org/abs/1508.07909
  - SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing:
    - https://arxiv.org/abs/1808.06226

Preparations for the following weeks

- **Gemini API Key:**
  - Use your existing google account or create a new free account
  - Go to https://aistudio.google.com/apikey
  - Get your API key and store it in a safe place
- **Create a HuggingFace account:**
  - https://huggingface.co/join
- **Set up Python & Jupyter environment:**
  - I use Python 3.12 for running code locally
  - For inference or fine-tuning requiring GPU, I use Google Colab (more information later)

References for further study

- Stanford CS229 I Machine Learning I Building Large Language Models (LLMs):
  - https://www.youtube.com/watch?v=9vM4p9NN0Ts
- Andrej Karpathy: Let's build GPT: from scratch, in code, spelled out:
  - https://www.youtube.com/watch?v=kCc8FmEb1nY
- Sebastian Raschka: https://magazine.sebastianraschka.com/
- Sebastian Raschka: Build a Large Language Model (From Scratch):
  - https://www.manning.com/books/build-a-large-language-model-from-scratch
  - https://github.com/rasbt/LLMs-from-scratch