Experiential Learning
University of Tehran
Instructors: Dr. Nadi

Project Report

An Overview of Language Models

— Fall 2024 —

Project Collaborators:

Name	**Student ID**
Mobin Roohi	610300060
Amirhossein Ghorbaninezhad	610300089
Vida Karbasi	610300094
Ali Ghozati	610300090

Abstract

This report provides an overview of language models, explaining their evolution from early statistical approaches to modern deep learning architectures such as transformers. It examines the fundamental principles underlying these models, including language representation and neural network-based learning, while discussing key challenges. This report also includes up to date information regarding the current large language model (LLM) landscape that has taken shape in the world. These model have risen as the primary source of excitement revolving around AI these days.

Contents

1 Machine Learning and Markov Model in Natural Language Processing

1.1 Hidden Markov Models

A Hidden Markov Model is a statistical model which is also used in Machine Learning. It can be used to describe the evolution of observable events that depend on internal factors, which are not directly observable. The applications where the HMM (Hidden Markov Model) can be used are cases such as time series data, audio and video data, and text data or Natural Language Processing data. In this part, our main focus is on those applications of NLP where we can use the HMM for better performance of the model, for example, we can use HMM in the Part-Of-Speech tagging.

1.2 What is POS-tagging?

We have learned that the part of speech indicates the function of any word, like what it means in any sentence. There are commonly nine parts of speeches; noun, pronoun, verb, adverb, article, adjective, preposition, conjunction, interjection, and a word need to be fit into the proper part of speech to make sense in the sentence. POS tagging is a very useful part of text preprocessing in NLP as we know that NLP is a task where we make a machine able to communicate with a human or with a different machine. So it becomes compulsory for a machine to understand the part of speech. Classifying words in their part of speech and providing their labels according to their part of speech is called part of speech tagging or POS tagging OR POST.

1.3 POS tagging with Hidden Markov Model

Let's take an example to make it more clear how HMM helps in selecting an accurate POS tag for a sentence. Consider the sentence ”The cat sat on the mat.”, We want to determine the most likely POS for each word and the HMM model.

Possible POS tags are: • Noun (N) • Verb (V) • Determiner (D) • Preposition (IN)

HMM calculation:

• ”The” is most likely a determiner (D)

• Given that ”The” is a D, ”cat” is likely a N

• Following a noun, ”sits” is likely a verb

• Similarly, ”on” is a prespoition (IN) and ”the” is again a determiner (D)

• Finally, ”mat” is most likely another noun (N)

Example:

Let's illustrate a Hidden Markov Model (HMM) example for Part-of-Speech (POS) tagging, a common NLP task. We'll keep it simple for demonstration.

The Scenario:

Imagine we have a simplified language with only two parts of speech: Noun (N) and Verb (V). We observe sentences like "The cat sat" and want to tag each word with its correct POS.

The HMM Components:

States (Hidden): These are the POS tags we want to predict (N, V). Observations (Visible): These are the words in the sentence ("The", "cat", "sat"). Transition Probabilities: The probability of moving from one state to another (e.g., the probability of a Noun following a Verb). We'll represent this as P(State$_t|State_t - 1)$.

Emission Probabilities: The probability of observing a particular word given a state (e.g., the probability of observing "cat" given the state Noun). We'll represent this as P(Observation$_t|State_t)$.

Initial Probabilities: The probability of starting in a particular state (e.g., the probability of the first word being a Noun). We'll represent this as P(State$_1)$.

Example Probabilities (Simplified):

Let's make up some probabilities for our example. In a real-world scenario, these would be learned from a large corpus of tagged text.

Initial Probabilities:

P(Noun) = 0.6 (We assume nouns are more likely to start a sentence)

P(Verb) = 0.4

Transition Probabilities:

P(Noun — Noun) = 0.4 (A noun is sometimes followed by another noun, like in "red car")

P(Verb — Noun) = 0.6 (A noun is often followed by a verb)

P(Noun — Verb) = 0.3 (A verb is less likely to be followed by a noun)

P(Verb — Verb) = 0.7 (Verbs can be followed by other verbs, e.g., "run fast")

Emission Probabilities:

P("The" — Noun) = 0.1 (The word "The" can sometimes be a noun, although unlikely)

P("The" — Verb) = 0.0 (The word "The" is almost never a verb)

P("cat" — Noun) = 0.8 (The word "cat" is very likely a noun)

P("cat" — Verb) = 0.05 (The word "cat" could be a verb in some rare context)

P("sat" — Noun) = 0.05 (The word "sat" can sometimes function as a noun)

P("sat" — Verb) = 0.9 (The word "sat" is very likely a verb)

Tagging the Sentence "The cat sat":

Now, how do we use this to tag the sentence? The Viterbi algorithm is commonly used for this. It finds the most likely sequence of hidden states (POS tags) given the observed words.

Here's a simplified breakdown of how Viterbi would approach it (without the full dynamic programming table):

Start: Calculate the probability of each possible first tag:

P(Noun — "The") = P("The" — Noun) * P(Noun) = 0.1 * 0.6 = 0.06 P(Verb — "The") = P("The" — Verb) * P(Verb) = 0.0 * 0.4 = 0.0 Noun is more likely for the first word.

Second Word ("cat"): For each possible tag of "cat", consider the most likely previous tag:

For Noun: P(Noun — "cat") = P("cat" — Noun) * max(P(Noun — "The"), P(Verb — "The")) * P(Noun — Noun or Verb). For Verb: P(Verb — "cat") = P("cat" — Verb) * max(P(Noun — "The"), P(Verb — "The")) * P(Verb — Noun or Verb). We'd calculate these and see which tag for "cat" is more probable.

Third Word ("sat"): We repeat the process, considering the most likely tag for "cat" when calculating the probabilities for "sat" being a Noun or a Verb.

Backtracking: After calculating probabilities for the last word, we backtrack through the highest probability paths to find the most likely sequence of tags.

Expected Output:

The Viterbi algorithm (or similar methods) would likely output:

"The" - Noun "cat" - Noun "sat" - Verb

Key Improvements over Simpler Methods:

HMMs handle ambiguity much better than simple lookup tables. For example, "run" can be a noun or a verb. The HMM considers the context (previous words and their tags) to determine the most likely POS.

1.4 Maximum Entropy in NLP

Maximum Entropy is a powerful statistical method frequently employed in Natural Language Processing for tasks like text classification, part-of-speech tagging, and named entity recognition. It's based the principle of maximizing uncertainty or entropy, subject to constraints derived from observed data.

1.5 Backpropagation in Neural Network

Backpropagation (Backward Propagation of Errors) is a method used to train artificial neural networks. Its goal is

to reduce the difference between the model's predicted output and the actual output by adjusting the weights in

the network.

What is Backpropagation?

Backpropagation is a powerful algorithm in deep learning, primarily used to train artificial neural networks. It

works iteratively, minimizing the cost function by adjusting weights and biases.

In each epoch, the model adapts these parameters, reducing loss by following the error gradient. Backpropagation

often utilizes optimization algorithms like gradient descent or stochastic gradient descent

1.6 Working of Backpropagation Algorithm

The Backpropagation algorithm involves two main steps: the **Forward Pass** and the **Backward Pass**

1.7 How does the Forward Pass Work?

In the forward pass, the input data is fed into the input layer. These inputs, combined with their respective

weights, are passed to hidden layers.

For example, in a network with two hidden layers (h1 and h2 as shown in next page), the output from h1 serves as

the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer applies an activation function like ReLU (Rectified Linear Unit), which returns the input if it's

positive and zero otherwise. This adds non-linearity, allowing the model to learn complex relationships in the data.

Finally, the outputs from the last hidden layer are passed to the output layer, where an activation function,

such as softmax, converts the weighted outputs into probabilities for classification.

1.8 How does the Backward Pass Work?

In the backward pass, the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the Mean Squared Error (MSE), given by:

$$MSE = (Predicted\ Output - Actual\ Output)^2 \tag{1}$$

Once the error is calculated, the network adjusts weights using gradients, which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer, ensuring that the network learns and improves its performance. The activation function, through its derivative, plays a crucial role in computing these gradients during backpropagation.

2 Formal Grammars in Natural Language Processing: A Foundation for Understanding Language Structure

Formal grammars are a cornerstone of Natural Language Processing (NLP), providing a precise and mathematical way to describe the structure of language. They offer a framework for understanding how words combine to form phrases, clauses, and sentences, enabling computers to parse and interpret human language. While natural language is often messy and defies strict rules, formal grammars provide valuable abstractions that capture key syntactic patterns and serve as the basis for many NLP applications.

2.1 The Basics of Formal Grammars:

A formal grammar, in its most basic form, consists of:

A set of terminals: These are the actual words or symbols that make up the language (e.g., ”cat,” ”the,” ”sat,” ”on,” ”mat”). A set of non-terminals: These are symbols that represent grammatical categories or syntactic constituents (e.g., ”Noun,” ”Verb,” ”Adjective,” ”Noun Phrase,” ”Verb Phrase,” ”Sentence”). A set of production rules: These rules define how non-terminals can be rewritten as combinations of terminals and/or other non-terminals. They express the grammatical relationships within the language. A common notation for production rules is A $\rightarrow$ B, where A is a non-terminal and B is a sequence of terminals and/or non-terminals. A start symbol: This is a special non-terminal that represents the top-level structure of the language, typically a ”Sentence” (S). A grammar generates a language by starting with the start symbol and repeatedly applying the production rules until only terminals remain. The set of all strings of terminals that can be derived in this way constitutes the language defined by the grammar.

Example:

Consider a simple grammar for a subset of English:

Terminals: the, cat, sat, on, mat

Non-terminals: S, NP, VP, N, V, P

Production rules:

S $\rightarrow$ NP VP

NP $\rightarrow$ Det N

NP $\rightarrow$ N

VP $\rightarrow$ V NP

VP $\rightarrow$ V P NP

Det $\rightarrow$ the

N $\rightarrow$ cat

N $\rightarrow$ mat

V $\rightarrow$ sat

P $\rightarrow$ on

Start symbol: S

Using this grammar, we can derive the sentence ”The cat sat on the mat” as follows:

S $\rightarrow$ NP VP

$\rightarrow$ Det N VP

$\rightarrow$ the N VP

$\rightarrow$ the cat VP

$\rightarrow$ the cat V P NP

$\rightarrow$ the cat sat P NP

$\rightarrow$ the cat sat on NP

$\rightarrow$ the cat sat on Det N

$\rightarrow$ the cat sat on the N

$\rightarrow$ the cat sat on the mat

2.2 Types of Formal Grammars:

Formal grammars are often classified according to the Chomsky hierarchy, which defines a hierarchy of grammar types based on the complexity of their production rules:

Regular Grammars (Type 3): These are the simplest type of grammar, where production rules are of the form A $\rightarrow$ aB or A $\rightarrow$ a, where A and B are non-terminals and ’a’ is a terminal. Regular grammars can be recognized by finite automata. They are often used for tasks like morphological analysis and tokenization. Context-Free Grammars (Type 2): In CFGs, production rules are of the form A $\rightarrow$, where A is a non-terminal and is a string of terminals and/or non-terminals. The left-hand side of the rule consists of a single non-terminal, and the right-hand side can be any combination of terminals and non-terminals. CFGs are powerful enough to capture many of the syntactic structures of natural language and are widely used in parsing. They can be recognized by pushdown automata. Context-Sensitive Grammars (Type 1): These grammars allow for more complex rules, including rules of the form A $\rightarrow$, where A is a non-terminal, and are strings of terminals and/or non-terminals (possibly empty), and is a non-empty string of terminals and/or non-terminals. The context in which A appears (and) can influence how it is rewritten. Context-sensitive grammars are more powerful than CFGs but are less commonly used in NLP due to their increased complexity. Recursively Enumerable Grammars (Type 0): These are the most general type of grammar, with no restrictions on the form of production rules. They can generate any language that can be recognized by a Turing machine. While theoretically powerful, they are rarely used in practice due to their extreme generality and the difficulty of parsing them. In NLP, Context-Free Grammars are the most widely used due to their balance between expressive power and computational tractability.

2.3 Formal Grammars in NLP Applications:

Formal grammars play a crucial role in various NLP tasks:

Parsing: Parsing is the process of analyzing a sentence according to a grammar to determine its syntactic structure. Parsers use formal grammars to build parse trees, which represent the hierarchical organization of the sentence. This information is essential for understanding the meaning of the sentence. Several parsing algorithms exist, including top-down parsing, bottom-up parsing, and chart parsing. Machine Translation: Formal grammars can be used to analyze the syntactic structure of the source language and generate the corresponding structure in the target language. This helps to ensure that the translated sentence is grammatically correct and preserves the meaning of the original sentence. Grammar Checking: Grammar checkers use formal grammars to identify grammatical errors in text. By comparing the structure of the input text to the rules of the grammar, they can detect violations of grammatical rules. Dialogue Systems: Formal grammars can be used to define the possible inputs that a dialogue system can understand. This allows the system to parse user input and respond appropriately. Information Extraction: Formal grammars can be used to identify specific entities and relationships in text. For example, they can be used to extract person names, locations, and dates from news articles.

2.4 Limitations and Extensions:

While formal grammars provide a valuable framework for understanding language structure, they also have limitations:

Ambiguity: Natural language is often ambiguous, meaning that a single sentence can have multiple possible interpretations. Formal grammars need to be able to handle ambiguity and provide all possible parses. Coverage: Creating a grammar that covers all the complexities and irregularities of natural language is a difficult task. Grammars often need to be extended and refined to handle new linguistic phenomena. Statistical Methods: Purely grammar-based approaches often struggle with the inherent variability and noise in natural language. Statistical methods, such as probabilistic context-free grammars (PCFGs), have been developed to address this issue by assigning probabilities to different parse trees. These probabilities can be learned from training data and used to select the most likely parse.

3 Automata

During the period from the first computer invented to the invention of internet, computers went from simple calculators to all-encompassing tools for processing and transferring information. This naturally raised the question, ”Just how much farther can they go?”.

This was the beginning of an endeavor in mathematics and engineering. While the engineer continuously asked, ”how to improve the machine further”, the mathematician asked, ”Can we answer these questions using math?”. And this was the start of a new field of research in mathematics which is currently known as ”Computer Science”.

A lot of models were made to abstractize the concept of an ”Automatic Computer”. Among them, one that is mainly focused on the science of language and logics, is the ”Automata”.

An automaton (automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.A particular type of an automaton which are more relevant to our topic of language models are ”Determenistic Finite Automata” or DFA for short.DFA is a simple and very basic automata, well suited for a small or limited finite number of states, inputs, and transition functions. Input can be at one state at a time, the state can be determined, and we know exactly the transition steps.

3.1 Formal Definition of DFA

A deterministic finite automaton is a quintuple $M = (K, \Sigma, \delta, S, F)$ where:

- K is a finite set of states,
- Σ is an alphabet,
- $S \in K$ is the initial state,
- $F \subseteq K$ is the set of final states, and
- δ is the transition function, a subset of $K \times \Sigma \to K$.

The language understood by a DFA are called ”natural languages”. For example we can design a simple DFA to accept all strings with a substring of 01.

4 Neural Language Model (2003)

The 2003 paper by Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, titled *”A Neural Probabilistic Language Model,”* represents a seminal contribution to the field of natural language processing (NLP) and the development of neural language models.This model tackles the “curse of dimensionality” by using neural networks to learn distributed representations of words. In this section, we will try to break down the key concepts from the paper and explain their significance.

4.1 Breaking New Ground

Before the introduction of neural networks to language modeling, traditional approaches relied heavily on n-grams and other statistical methods to predict the probability of a sequence of words. These methods, while effective to an extent, faced significant limitations such as data sparsity and the ”curse of dimensionality”. The curse of dimensionality refers to the exponential growth of possible word sequences as the length of the sequence increases. For example, modeling the joint distribution of 10 consecutive words from a vocabulary of 100,000 words involves $100,000^{10}$ potential sequences, making traditional methods impractical.

Traditional n-gram models reduce this by using short, overlapping sequences, but they struggle with new sequences not seen in training. Bengio’s model addresses this by learning a continuous, distributed representation for words, enabling generalization to new sequences.

4.2 Word Embeddings and Distributed Representations

One of the most significant contributions of the 2003 paper was the concept of word embeddings. In this model, words are represented as vectors in a continuous, high-dimensional space. This allows the model to capture semantic relationships between words more effectively. For instance, in the embedding space, the distance between vectors can reflect the similarity or dissimilarity between words. This representation has become a cornerstone of modern NLP, enabling more nuanced understanding and generation of human language by machines.

4.3 Neural Network Architecture

The neural language model proposed by Bengio et al. utilized a feedforward neural network to predict the probability of a word given its preceding context. This architecture consisted of an input layer, a hidden layer, and an output layer. The hidden layer enabled the model to learn complex, non-linear relationships between words, significantly improving the accuracy of language modeling compared to traditional statistical methods.

4.4 Impact on Large Language Models

The principles established by Bengio et al. have been integral to the evolution of LLMs. Modern models, such as GPT-3 by OpenAI, BERT by Google, and other transformer-based architectures, build upon the idea of word embeddings and neural networks to handle vast amounts of text data. These models leverage attention mechanisms and deep learning techniques to achieve state-of-the-art performance in various NLP tasks, from translation to summarization to question-answering.

Moreover, the shift from discrete, symbolic representations of language to continuous, distributed representations has enabled these models to generalize better and understand context more deeply. This paradigm shift, first conceptualized in the 2003 paper, has propelled advancements in AI, making possible the sophisticated language understanding and generation capabilities seen today.

5 Introduction to Natural Language Processing (NLP)

Natural Language Processing (NLP) focuses on the interaction between computers and human language. Its applications span a wide range of domains, including search engines, virtual assistants, and social media analysis.

5.1 Defining NLP

NLP is the field of artificial intelligence that equips machines with the ability to read, interpret, and generate human language. It bridges linguistics and computer science to enable:

- Machine translation.

- Text summarization.
- Sentiment analysis.
- Speech recognition.

5.2 Key Challenges

1. **Ambiguity:** Words or sentences can have multiple interpretations.
2. **Data Scarcity:** Many languages and domains lack annotated datasets.
3. **Dynamic Context:** Meaning can shift depending on context, requiring models to adapt.

5.3 Applications of NLP

- **Search and Retrieval:** Google uses NLP to improve query results.
- **Chatbots:** Virtual assistants like Alexa and Siri rely on NLP.
- **Content Moderation:** Automatically flagging harmful or inappropriate content.
- **Machine Translation:** Services like Google Translate facilitate communication across languages.

5.4 Traditional Approaches in NLP

Before the advent of deep learning, NLP relied heavily on statistical models and hand-engineered features.

5.5 N-grams for Language Modeling

An n-gram is a sequence of n words used to estimate the likelihood of word sequences:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_{i-n+1}).$$

For example, in trigrams ($n = 3$):

$$P(\text{sentence}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2).$$

5.6 Bag-of-Words (BoW)

BoW represents text by counting word frequencies, disregarding order:

$$\text{Document: "cats chase mice"} \rightarrow [1, 0, 1, 1].$$

While simple, this method loses syntactic relationships between words.

5.7 One-Hot Encoding

Each word in a vocabulary is represented by a binary vector:

$$\text{Vocabulary: [''cat'', ''dog'', ''fish''],} \quad \text{cat} \rightarrow [1, 0, 0].$$

This approach leads to sparsity and lacks semantic relationships.

6 Deep Learning for NLP

Deep learning revolutionized NLP by introducing models capable of automatically learning features from data.

6.1 Neural Networks Overview

Neural networks consist of interconnected layers:

- **Input Layer:** Processes raw data (e.g., word embeddings).
- **Hidden Layers:** Transform inputs through weights and activation functions.
- **Output Layer:** Produces predictions (e.g., next word in a sequence).

6.2 Recurrent Neural Networks (RNNs)

RNNs model sequential data by maintaining hidden states across time:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t),$$

where W_{hh} and W_{xh} are weight matrices. Variants include:

- **LSTM:** Introduces gates to control information flow and address long-term dependencies.
- **GRU:** Simplifies the LSTM architecture while retaining performance.

6.3 Applications of Neural Networks in NLP

- **Text Classification:** Labeling documents as spam or non-spam.
- **Machine Translation:** Converting text between languages.
- **Sentiment Analysis:** Extracting emotion from text.

7 Word Embeddings

Word embeddings map words to dense, low-dimensional vectors in a continuous space.

7.1 Motivation and Concept

Word embeddings capture semantic meaning:

$$\text{king} - \text{man} + \text{woman} = \text{queen}.$$

This enables models to generalize across tasks, unlike one-hot encoding.

7.2 Training Techniques

7.2.1 Word2Vec

Word2Vec employs two architectures:

- **CBOW:** Predicts a word given its context.
- **Skip-Gram:** Predicts context words given a target word.

7.2.2 GloVe

GloVe captures global co-occurrence statistics to produce embeddings that reflect word relationships.

7.2.3 FastText

FastText represents words as character n-grams, handling out-of-vocabulary words effectively.

7.3 Advanced Embedding Models

- **ELMo:** Context-sensitive embeddings derived from a bidirectional language model.
- **BERT:** Generates contextualized embeddings by processing text bidirectionally.
- **Transformers:** Architectures like GPT enable dynamic context handling.

7.4 Applications of Word Embeddings

- **Semantic Similarity:** Identifying related words.
- **Named Entity Recognition (NER):** Extracting entities like names or dates.
- **Text Summarization:** Generating concise text summaries.

7.5 Limitations of Word Embeddings

- **Biases:** Gender and societal biases can propagate into embeddings.
- **Context Insensitivity:** Static embeddings fail to capture varying meanings of words.

8 Mathematical Foundations of Word Embeddings

8.1 Distributional Semantics

The distributional hypothesis states:

”Words used in similar contexts have similar meanings.”

This underpins the creation of word embeddings.

8.2 Objective Functions

Word2Vec minimizes the negative log-likelihood of predicting context words:

$$J = -\sum_{w \in C} \log P(w \mid t),$$

where $P(w \mid t)$ is the probability of a context word w given target t.

8.3 Similarity Metrics

Common metrics for comparing word embeddings:

- **Cosine Similarity:**
$$\text{sim}(u, v) = \frac{u \cdot v}{\|u\|\|v\|}.$$
- **Euclidean Distance:**
$$d(u, v) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}.$$

9 Transformer Networks

9.1 Encoder-Decoder Architecture

A common architecture in sequence-to-sequence models is the encoder-decoder architecture. In this architecture, generally, the model consists of an encoder component, which receives the input sequence and learns to encode it into complex representations. Then, a decoder component uses these representations from the encoder to generate the output sequence. Here, the decoder essentially acts as a conditional language model, taking in the encoded input from the encoder and the leftwards context of the target sequence and predicting the subsequent target sequence.

Here is a diagram showing the encoder-decoder architecture:

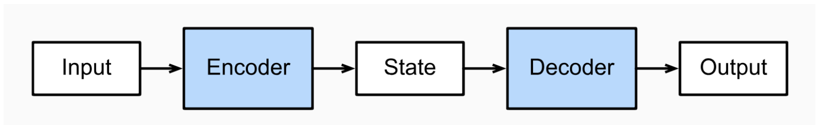

Figure 1: Encoder-Decoder Architecture

Encoder-decoder architectures can successfully be employed in sequence to sequence tasks with varying input and output sizes, such as machine translation, text summarization, and more.

[Dive Into Deep Learning]

9.2 Limitations of Recurrent Networks

Recurrent neural networks [Reference], long short-term memory [Reference], gated recurrent [Reference], and similar recurrent networks, have been firmly established well-performing approaches in sequence modeling and transduction problems such as language modeling, machine translation, and other NLP problems.

Recurrent models typically compute along the symbol positions of the input and output sequences. This means that they perform their computations sequentially. They learn a sequence of hidden states h_t as a function of the previous hidden state h_{t-1} and the input for position t. This sequential nature of processing limits parallelization within training examples, which becomes a problem with longer sequences as memory constraints limit batching across examples and training times increase.

Moreover, these models can not quite capture the long-term dependencies and contexts of longer sequences. To mitigate these problems and improve model performance, a ground-breaking architecture called the Transformer Network was proposed in 2017 [Transformer]. The Transformer uses the following mechanisms in its architecture:

1. Self-Attention
2. Multi-Head Attention
3. Positional Encoding
4. Batch Norm
5. Residual Connections
6. Masked Multi-Head Attention

We will first discuss some of these concepts briefly before providing the full Transformer architecture.

9.3 Mechanisms in Transformer

9.3.1 Self-Attention

To build up to the Transformer network, we need to first speak about its most fundamental component: **Self-Attention**.

Static Embeddings, as previously discussed, convert a token into a feature vector embedding that represents the token with a vector of numbers. Static embeddings, unfortunately, do not take the context of the token into account when creating the vector embedding. This is a limitation of the static embeddings learned using algorithms such as seq2vec.

To expand the idea of static embeddings to include contextualized information, self-attention is used to create contextualized embeddings that more adequately represent tokens in a sentence.

Self-attention inputs the input tokens X and then uses three learnable weight matrices,

$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$$
$$W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$
$$W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

to obtain the **query**, **key**, and **value** matrices,

$$Q = XW^Q$$
$$K = XW^K$$
$$V = XW^V$$

which we will use to obtain $A(Q, K, V)$ contextualized embedding. The final equation for self-attention using these values is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

9.3.2 Positional Encoding

Another component of the Transformers is the **Positional Encoding**. Positional encodings are additional encodings that get added to the input encoding to inject positional information into self-attention. The reason for this is that self-attention on its own does not distinguish along positions and does not content itself with time-steps, therefore it would perform poorly with regards to learning positional patterns. The positional encoding provides the positional information needed for the model to achieve better spatial clarity and performance.

Many possible functions can be used for positional encoding, however, they must all have the following properties:

1. **Not Ambiguous** - Have different values for different positions.

2. **Deterministic** - It needs to be able to be calculated using the position values deterministically for the model to learn the position patterns.

3. **Distance Encoded** - The encodings need to contain distance information; for instance, the encoding should allow the model to learn that position 1 is as far away as position 11 from position 6.

4. **Work with sequences longer than encountered before** - It needs to work for sequences far longer than the sequences it is trained on.

The positional encoding provided in the Transformer paper is the **sinusoidal positional encoding**.

The sinusoidal positional encoding assigns each position p in the sequence a vector $PE(p)$ of the same dimension d_{model} as the embedding vectors. This encoding is defined as:

$$\begin{aligned} PE(p, 2i) &= \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right) \\ PE(p, 2i+1) &= \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right) \end{aligned}$$

where p is the position in the sequence, i is the dimension index, and d_{model} is the dimensionality of the encoding and token embeddings.

After obtaining the desired positional encodings (PEs), we have two options: either concatenate them with the input embeddings, which increases computational cost, or follow the approach suggested in the Transformer paper—adding the PEs directly to the input embeddings and allowing the model to learn their integration effectively.

$$\tilde{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{p}_t$$

9.3.3 Multi-Head Attenion

As stated before, Transformers offer massive parallelization benefits due to their extensive use of attention. To facilitate this, multiple heads are performing attention at once. This is called the **Multi-Head Attention**. Here is its equation:

$$\begin{aligned} &\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{h}\right) W^O \\ &\text{where } \text{head}_\text{i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

The process of multi-head attention can be observed in the following diagram.

In the original work, the authors chose $h = 8$ attention heads. One more important point is that in multi-head attention, usually the Q, K, and V matrices are given to it as input, and there is no need to compute it, unlike in self-attention. However, if they are not provided in the input, they can be taken to be equal to X.

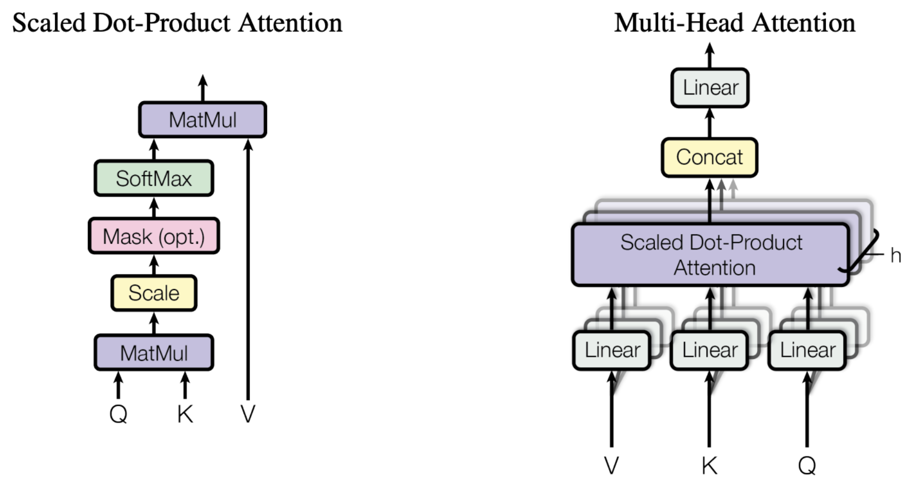

Figure 2: Multi-Head Attention consists of several attention layers running in parallel.

9.4 Transformer Architecture (2017)

Using the components discussed in the previous section, we can build the Transformer according to the encoder-decoder architecture as seen in figure 3. In the original work, 6 encoder layers and 6 decoder layers were trained.

9.5 Impact of Transformers

Transformers were able to achieve state-of-the-art performance compared to the recurrent models. Ever since their introduction in 2017, Transformers have revolutionized the field of natural language processing and machine learning as a whole.

Transformers became the foundation for a wave of groundbreaking models, such as BERT, GPT, and T5, which pushed the boundaries of tasks like translation, summarization, question answering, and text generation. The architecture also extended beyond NLP, impacting areas such as computer vision (e.g., Vision Transformers), time-series analysis, and even protein structure prediction (e.g., AlphaFold [Reference]). The introduction of pre-training and fine-tuning paradigms further solidified their impact, making domain adaptation efficient and accessible.

10 Pre-Trained Models and Fine-Tuning

Given the previous overview on Transformers, two fundamental concepts in NLP are pre-training and fine-tuning language models such as the Transformer. In most of the state-of-the-art language models seen today, these two concepts are used.

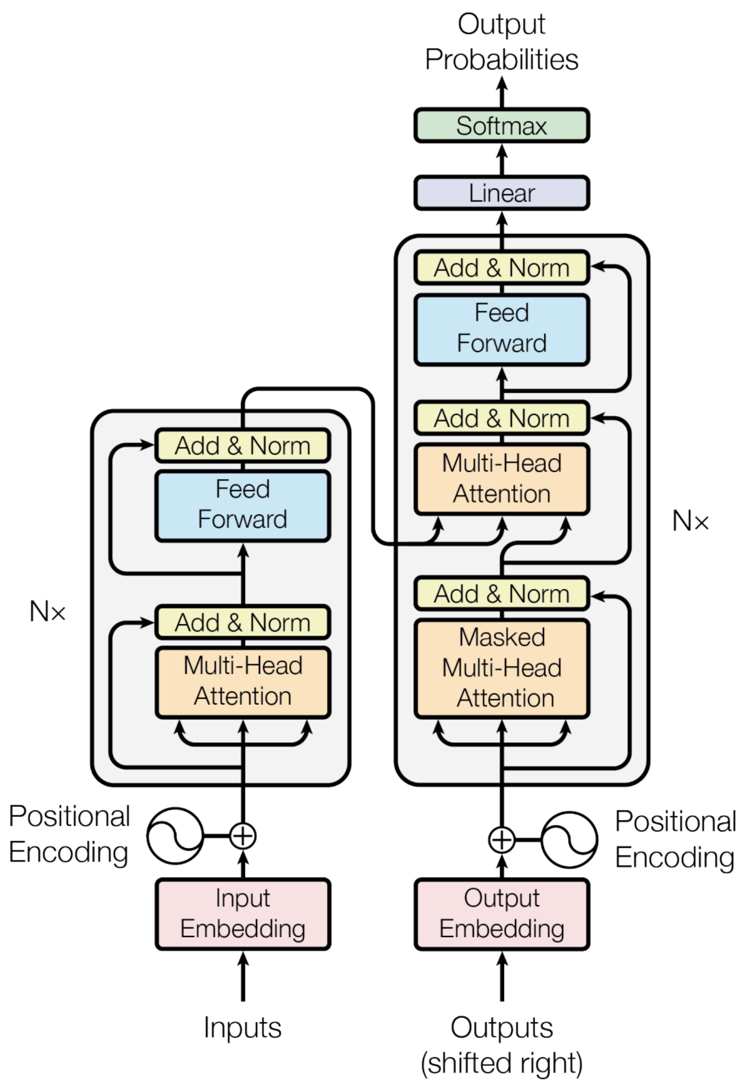

Figure 3: Transformer Architecture

10.1 Pre-Training

Pre-training is the process of training a model on a large corpus of unlabeled data to learn a general-purpose understanding and representation of the language. This step typically uses self-supervised learning, where the model is trained to predict parts of the input based on the rest, without requiring manually labeled datasets.

Some tasks used for the self-supervised learning in pre-training include:

1. **Masked Language Modeling (MLM):** In MLM, some input tokens are randomly masked, and the model is trained to predict these tokens based on their context. This task was used in the training of BERT[1] model by Google in 2018 [Reference].
2. **Causal Language Modeling (CLM):** The model predicts the next token in a sequence given its previous tokens, operating in an autoregressive[2] fashion. GPT[3] (2018) by OpenAI uses this approach, training models left-to-right.

Pre-training allows for the learning of syntactic and semantic patterns of the language. These pre-trained models are then used as a foundation, significantly reducing the labeled data requirements for downstream tasks. In the larger pre-trained models improvement of performance can be achieve on diverse tasks.

10.2 Fine-Tuning

After pre-training is complete **Fine-tuning** is performed on a pre-trained model to purpose it for a specific task using labeled data. This step typically modifies the pre-trained weights slightly to optimize performance for the target task.

Transitioning from the pre-training and fine-tuning paradigms, we will explore BERT and GPT which are successful examples of using transformers as pre-trained models and then fine-tuning them to achieve great performance across NLP tasks.

10.3 GPT (2018)

GPT (Generative Pre-Trained Transformer) as the names suggest was a pre-trained model based on the decoder of the transformer introduced in 2018. It introduced unsupervised pre-training followed by supervised fine-tuning. This model uses a unidirectional next word prediction task for training the model on multiple stacks of transformer decoders.

This pre-trained could then be fine-tuned by an output layer or component to specific NLP tasks such as question answering, translation, or text classification.

GPT was trained on the BooksCorpus dataset (a collection of about 11,000 unpublished books) and other publicly available datasets.

[1]Bidirectional encoder representations from transformers
[2]Left-to-right
[3]Generative pre-trained transformer

Here are some of the specifications of the GPT:

Specification	**GPT (2018)**
Number of Transformer Layers	12
Number of Parameters	117M
Number of Multi-Head Attentions	12

Table 1: Specifications of GPT

The GPT achieved state-of-the-art performance in many tasks such as commonsense reasoning and question answering.

10.4 BERT (2018)

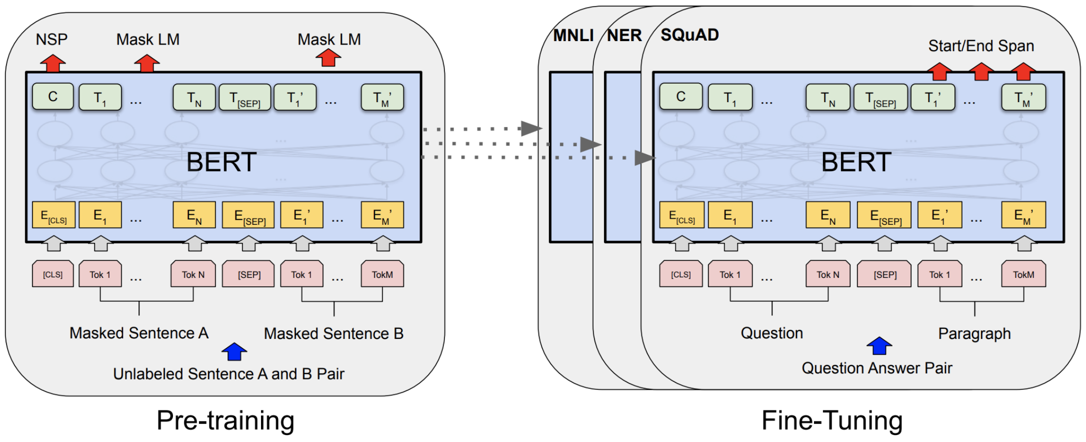

Figure 4: BERT Training

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model introduced by Google in 2018. Unlike GPT, which is an autoregressive model generating text left-to-right, BERT uses bidirectional language representation by considering both left and right contexts. This bidirectional contextual understanding allows BERT to excel at tasks requiring a comprehensive understanding of the input text, such as classification and question answering.

BERT was pre-trained using only the encoder layer of transformer. One of the innovations of the BERT paper was the masked language model task which allows the model to learn bidirectionally by masking 15% of the input tokens, either from left or right and asking the model to predict it.

Additionally, next sentence prediction task was also leveraged for model training. This task allows the model to understand the relationship between two sentences. The model is fed pairs of sentences and must predict whether the second sentence is the actual next sentence in the original text. This is particularly useful for tasks like question answering and natural language inference.

There were to versions of BERT released, BERT Base and BERT Large. Here are some specification relating to these two pre-trained models:

Specification	**BERT-Base**	**BERT-Large**
Number of Transformer Layers	12	24
Number of Parameters	110M	340M
Number of Multi-Head Attentions	12	16

Table 2: Specifications of BERT-Base and BERT-Large

After pre-training, BERT can be fine-tuned for specific NLP tasks by adding a simple output layer. Fine-tuning adjusts the parameters of BERT slightly to adapt it to the specific task, such as text classification, question answering, named entity recognition, and more.

11 GPT-2 and GPT-3

11.1 GPT 2 (2019)

In 2019 GPT-2 was introduced. GPT-2 was step up from GPT in terms of its scale. It used a very similar architecture to GPT, however it was much larger which allowed it to have boosted performance across different tasks. It kept the transformer decoder-based self-supervised style of training a pre-trained model and supervised fine-tuning.

It was trained on a dataset of approximately 8 million web pages scraped from the internet (WebText) which was about 40 GB of data.

11.2 GPT-3 (2020)

GPT-3 was another state-of-the-art language model developed by OpenAI and released in 2020. It represented a significant advancement over its predecessor, GPT-2, in terms of scale, capabilities, and versatility. It kept the same architecture but yet again increased scale and improved learning methodology.

The model was pre-trained on a massive and diverse dataset of internet text which was about 570 GB large.

The following table shows the specifications of GPT 1 through GPT 3 compared against one another.

Feature	**GPT-1 (2018)**	**GPT-2 (2019)**	**GPT-3 (2020)**
Layers	12	48	96
Parameters	117M	1.5B	175B
Hidden Size	768	1600	12,288
Attention Heads	12	25	96
Vocabulary Size	40,000	50,257	50,257
Context Length	512	1024	2048

Table 3: Specifications of GPT-1, GPT-2, and GPT-3

11.3 Zero-Shot and Few-Shot Learning

Zero-shot learning is paradigm where a model generalizes to new tasks or classes without seeing any labeled examples for those tasks during training.

Few-shot learning on the other hand is an approach where an ML model learns to perform a task or classify new classes with only a few labeled examples per class. It leverages prior knowledge from large-scale pretraining or meta-learning to adapt quickly to data-scarce scenarios.

While few-shot learning has been explored conceptually and in limited applications before, the practical realization and broad applicability of few-shot learning in NLP truly started with OpenAI's GPT-3 in 2020. GPT-3 displayed the ability to perform a wide range of tasks by using few-shot prompts, stopping the need for task-specific fine-tuning and changing how language models are used in practice. GPT-3 was capable of this due to its massive size and long training which resulted in a pre-trained model that has learned language representations well enough to be able to perform few-shot learning adequately. In some tasks, few shot learning on GPT-3 performed similarly to models fine-tuned to perform that task. GPT-3 also showed capability of performing zero-shot learning as well.

As an example of few-shot prompts consider the following input to a pre-trained large language model (4-shot prompt):

```
Text:  (lawrence bounces) all over the stage, dancing, running, sweating,
mopping his face and generally displaying the wacky talent that brought him
fame in the first place.
Sentiment:  positive

Text:  despite all evidence to the contrary, this clunker has somehow managed
to pose as an actual feature movie, the kind that charges full admission
and gets hyped on tv and purports to amuse small children and ostensible
adults.
Sentiment:  negative
```

```
Text:  for the first time in years, de niro digs deep emotionally, perhaps
because he's been stirred by the powerful work of his co-stars.
Sentiment:  positive

Text:  i'll bet the video game is a lot more fun than the film.
Sentiment:
```

---

12 Modern Large Language Models (2020 – Present)

12.1 LLMs

GPT-3 is an example of a **large language model**. It is considered "large" due to its massive amount of parameters (about 175 billion).

To formulate what constitutes an LLM[4], we can state some properties LLMs generally have:

1. **They have tens of billions of parameters**, e.g. 1.76 trillion parameters on OpenAI's GPT-4.
2. **They are trained on a massive and diverse corpus of text**, e.g. Meta's LLaMA was trained on about 1.4 TBs of text.
3. **They are capable of generalization such as zero-shot and few-shot learning**.
4. **They often support a large context window.**
5. **They require significant computational resources for training and inference**, e.g. Google's Gemini Ultra cost about 190 million dollars to train.

Now, we will briefly mention some of the modern LLMs that have taken the world by storm. It will become apparent that most LLMs today are trained in the industry rather than in academia.

12.2 OpenAI's GPT-3.5 (2022)

GPT-3.5 marked a turning point in making large language models broadly accessible. Originally used as the backbone for ChatGPT when it launched in late 2022, GPT-3.5's conversational fluency and general-purpose language understanding impressed millions of users who had never before interacted with AI at such scale. Its underlying architecture was an evolution of GPT-3, the refinement in dialogue management and context awareness made it a favorite for applications ranging from customer support to creative writing. This model's success helped fuel a wave of public enthusiasm and research investment in conversational AI technologies.

[4]Large language model

12.3 OpenAI’s GPT-4 (2023)

Launched in March 2023, GPT-4 took the capabilities of its predecessor to a new level by incorporating multimodal inputs and improved reasoning skills. It expanded context window and the ability to process both text and images. Its performance improvements on professional exams and coding tasks caused both scientific admiration and commercial hype. GPT-4’s integration into products like ChatGPT Plus and Bing Chat underscored its market impact and solidified OpenAI’s leadership in generative AI research.

12.4 Google’s Gemini (2023)

Google’s Gemini released in 2023 as a rival in the race for multimodal intelligence. Designed to handle text, images, and even video, Gemini drew significant attention for its extended context lengths—up to millions of tokens—and robust integration of diverse data types. Gemini’s launch signaled Google’s determination to lead in next-generation AI systems and sparked widespread debate about the future interplay between AI multimodality and practical application. Although a very powerful model, the market was mostly dominated by OpenAI’s models due to their better overall performance and quality.

12.5 OpenAI’s GPT-4o (2024)

Introduced in May 2024, GPT-4o (with “o” standing for “omni”) pushed the envelope further by unifying text, image, and audio processing in a single model. Its design emphasized cost-efficiency and speed while delivering state-of-the-art performance on multilingual and multimodal benchmarks. The model quickly gained hype as it let more natural, humanlike interactions. By significantly reducing the cost per token compared to earlier models, GPT-4o demonstrated that high-performance multimodal AI could be both powerful and economically viable.

12.6 OpenAI’s o1 (2024)

Released in late 2024, OpenAI’s o1 model was a breakthrough in advancing AI reasoning. Unlike its prediction-only predecessors, o1 was designed to “think” through complex queries using reinforcement learning techniques that allowed it to explore multiple solution paths before giving an answer. This model excelled at challenging tasks in mathematics, coding, and scientific problem solving.

12.7 Deepseek R1 (2025)

Deepseek R1, released in January 2025 by a Chinese AI startup, has been regarded as a disruptive force in the AI landscape. This model achieved performance levels comparable to OpenAI’s advanced o1 series while costing a fraction of the price. This achievement has been regarded as an “AI Sputnik moment.” More importantly Deep Seek R1 is an open-source model unlike its OpenAI counterparts. Its release has caused intense discussion about international competitiveness in AI research. By demonstrating that reasoning and natural language understanding can be achieved with less costly training, Deepseek R1 has pushed scientific boundaries and reshaped market expectations for future AI development.

The rise of powerful modern LLMs such as GPT-4-o and Google Bard has led many to believe we have already achieved ”machine consciousness”. But is this true? If it’s true, how would we know it? If it’s not, how far are we from it?

13 Are Modern LLMs Conscious?

Instead of a philosophical discussion about what ”consciousness” even is, we ask a much simpler question: ”Are modern LLMs capable of reasoning?” This question has its fair share of both concurring and opposing opinions among academics. Although most academics believe that LLMs are not yet capable of reasoning.

But if they are not reasoning, what are they doing? In *Can Large Language Models Reason* (source), Mitchell answers this question:

> ”If it turns out that LLMs are not reasoning to solve the problems we give them, how else could they be solving them? Several researchers have shown that LLMs are substantially better at solving problems that involve terms or concepts that appear more frequently in their training data, leading to the hypothesis that LLMs do not perform robust abstract reasoning to solve problems but instead solve problems (at least in part) by identifying patterns in their training data that match, or are similar to, or are otherwise related to the text of the prompts they are given.”

This is not cognition; it’s merely mechanical perception.

> ”Some GPT-based LLMs (pre-trained on a known corpus) were much better at arithmetic problems that involved numbers that appeared frequently in the pre-training corpus than those that appeared less frequently. These models appear to lack a general ability for arithmetic but instead rely on a kind of ”memorization”—matching patterns of text they have seen in pre-training. As a stark example of this, Horace He, an undergraduate researcher at Cornell, posted on Twitter that on a dataset of programming challenges, GPT-3 solved 10 out of 10 problems that had been published before 2021 (GPT-3’s pre-training cutoff date) and zero out of 10 problems that had been published after 2021. GPT-3’s success on the pre-2021 challenges thus seems to be due to memorizing problems seen in its training data rather than reasoning about the problems from scratch.”

This is quite an indictment of GPT’s problem-solving capabilities. However, there is a vigorous debate about what exactly LLMs “understand” and how different it is from how humans understand. On the one hand, most academics hold (source) that models trained on language “will never approximate human intelligence, even if trained from now until the heat death of the universe.” Not all researchers agree, claiming that “the behavior of LLMs arises not from grasping the meaning of language but rather from learning complex patterns of statistical associations among words and phrases in training data and later performing ‘approximate retrieval’ of these patterns and applying them to new queries.” So they might not be capable of ”human reasoning” but they can be called capable of using ”machine reasoning”.

13.1 Can machines achieve human reasoning and understanding?

There are a multitude of challenges. First and foremost, how will we see these technologies understand our world? Second, when will we have the tools to know how they can?

OpenAI disclosed that GPT-4 scored very well on the Uniform Bar Exam, the Graduate Record Exam, and several high-school Advanced Placement tests, among other standardized exams to assess language understanding, coding ability, and other capabilities, but evidence of human-level intelligence in GPT-4 is sketchy.

Critics claim that data contamination was at play. People taking standardized tests answer questions they have not seen before, but a system like GPT-4 may have very well seen them in the training data. OpenAI claims to use a ”Substring Match” technique to search training data and tags for similar but not exact matches. OpenAI's method was criticized in one analysis as "superficial and sloppy." The same critics noted that "for one of the coding benchmarks, GPT-4's performance on problems published before 2021 was substantially better than on problems published after 2021—GPT-4's training cutoff. This is a strong indication that the earlier problems were in GPT-4's training data. There's a reasonable possibility that OpenAI's other benchmarks suffered similar contamination."

Shortcut Learning - ML and deep learning can cause unpredictable errors when facing situations that differ from the training data. This is because such systems are susceptible to shortcut learning; statistical associations in the training data allow the model to produce correct answers for the wrong reasons. Machine learning, neural nets, and deep learning do not teach concepts; instead, they teach shortcuts to connect responses to the training set and apply statistical associations and probability assumptions to produce correct answers without cognition of the intended query. Another study showed that "an AI system that attained human-level performance on a benchmark for assessing reasoning abilities relied on the fact that the correct answers were (unintentionally) more likely statistically to contain certain keywords. For example, answer choices containing the word 'not' were more likely to be correct."

So whenever we might think that LLMs have achieved a level where they are able to solve a unique problem, it might be that there was another very similar problem in their vast amount of database. Or it might be that they came to the correct answer due to wrong reasons.

Basically, the problem of how to appropriately benchmark the level of human intelligence is still an open problem. But the most widely known criteria for assessing human intelligence is the ”Turing Test."

The Turing test, originally called the imitation game by Alan Turing in 1949, is a test of a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human. Turing proposed that a human evaluator would judge natural language conversations between a human and a machine designed to generate human-like responses. The evaluator would be aware that one of the two partners in conversation was a machine, and all participants would be separated from one another. If the evaluator could not reliably tell the machine from the human, the machine would be said to have passed the test.

This still might not be able to assess whether machines are capable of human reasoning. But to some extent, this can attest that they have achieved some level of human understanding.

13.2 What the future holds—Bridging the gap between A.I and humans—

So far until now, the improvement of LLMs is mainly attributed to drastically increasing the size of Language Models. But it is unlikely that more improvement can be achieved in this regard and some major changes in architecture are needed. This has led many to analyze the process of the human mind in hopes of some motivation. We will explore some of these ideas.

14 Multimodal Models

Nowadays we have achieved good results in approximating human information processing in all five of the human senses. In the strive towards AGI, one might suggest that ”why not make a robot who can understand physical information like humans and then let it learn like humans” and the academics’ answer to that is ”why not?”. This is the main idea behind a trend in LLMs to integrate already existing models for computer vision, image processing, etc., into LLMs.

Multidirectional Models

A major difference between neural networks and the human mind is the fact that signals in neural networks travel from only one side to another. But the case of the human mind is not so simple. Multidirectional models are a way to mimic that sort of behavior in artificial neural networks.

Hybrid Models

There’s growing interest in creating hybrid AI models that combine the best of both worlds:

- Neural networks (like LLMs) for pattern recognition and dealing with unstructured data.
- Symbolic reasoning systems for applying formal logic, rules, and consistent deductions.

These hybrid models could bridge the gap between the probabilistic reasoning of LLMs and the symbolic, rule-based reasoning that humans excel at. This approach could allow AI systems to handle a wider variety of tasks with greater accuracy, from understanding natural language to performing complex logical deductions in fields like mathematics, law, or scientific reasoning.