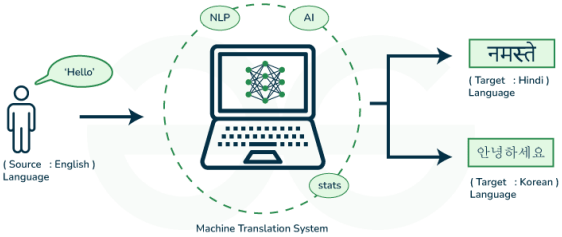
Natural Language Processing

- Language is central to human interaction; many of our daily activities revolve around text and language.
- Natural Language Processing (NLP) enables computers to understand and generate human language.

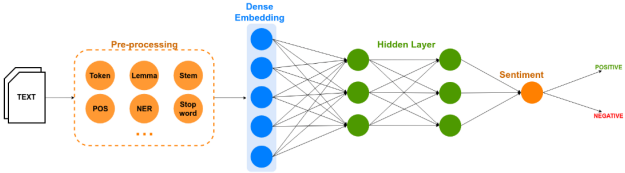
Translation



- NLP helps translate text from one language to another.

Figure adapted from geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/

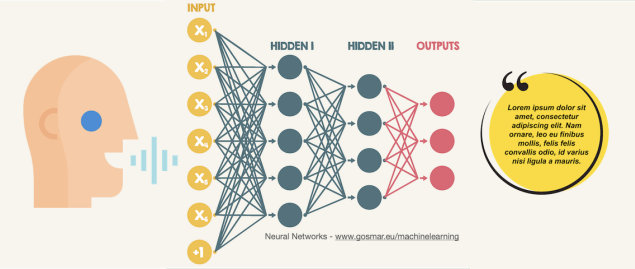
Sentiment Analysis



- Determines the sentiment (e.g., positive or negative) expressed in a text.

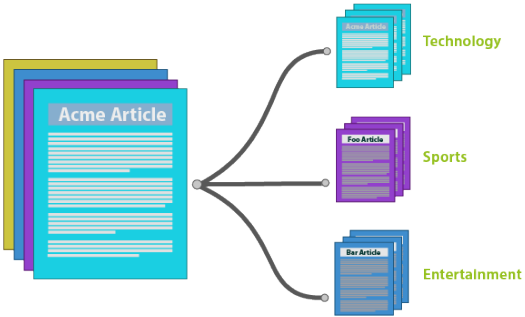
Figure adapted from www.mdpi.com/2079-9292/9/3/483

Speech Recognition



- Converts spoken language into text.

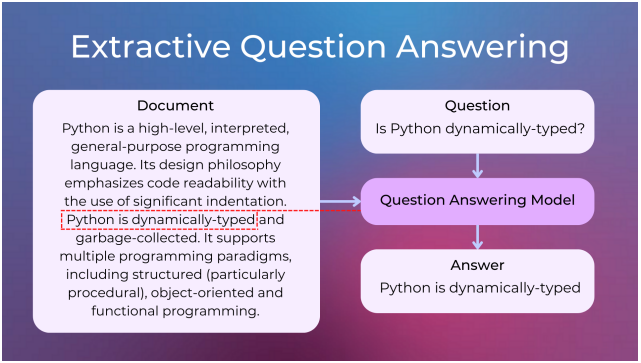
Text Classification



- Categorizes text into predefined groups or topics.

Figure adapted from towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

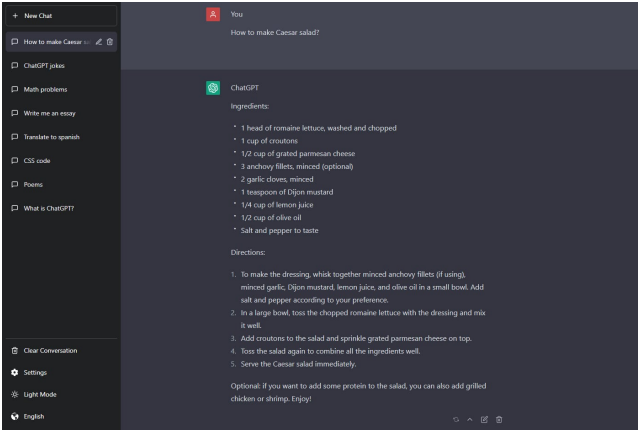
Question Answering



- Answers questions based on a given text or dataset.

Figure adapted from nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

Chatbots and Dialogue Systems



- NLP powers chatbots that can interact with users through text or speech.

Figure adapted from chatgpt.org/

Motivation and One-Hot Encoding

- Traditional models like one-hot encoding lack semantic understanding and fail to capture word relationships.
- One-hot encoding represents words as binary vectors where only one position is 1, making the representation sparse and non-semantic.
- We need dense, semantic word representations to improve natural language processing tasks.

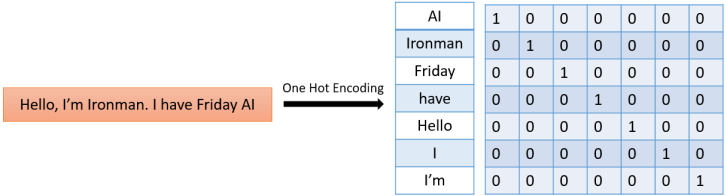
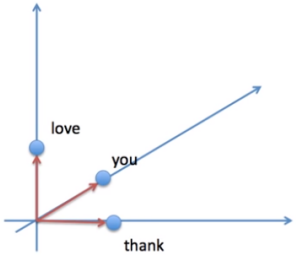


Figure adapted from medium.com/@kalyan45/natural-language-processing-one-hot-encoding-5b31f76b09a0

Example: One-Hot Encoding for Different Words

	thank	you	love
thank	1	0	0
you	0	1	0
love	0	0	1

unique word	encoding
thank	[1, 0, 0]
you	[0, 1, 0]
love	[0, 0, 1]



- Different words can be perpendicular to each other in space regardless of their meanings.

Figure adapted from andreaperlato.com/theorypost/introduction-to-word2vec/

Example: Word Embedding

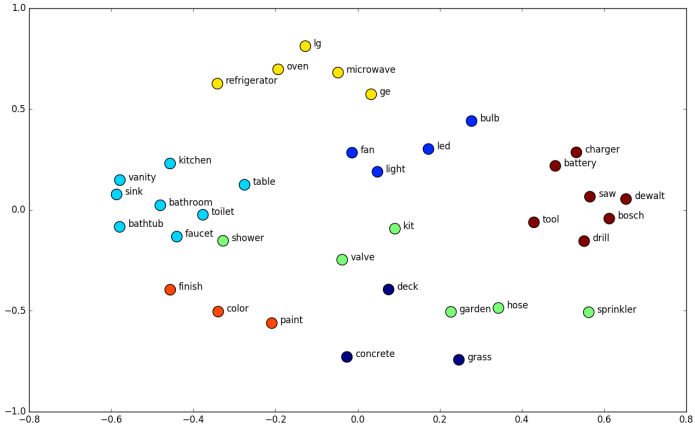


Figure adapted from classes.engr.oregonstate.edu/eecs/fall2023/ai534-400/unit4/word_embeddings.html

Skip-gram Example

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot
3	[The **wide** road shimmered in] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[The wide road **shimmered** in the hot] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [in the hot **sun**].	sun, in sun, the sun, hot

Different window sizes and samples drawn from context words and their target

Figure adapted from tensorflow.org/text/tutorials/word2vec

Word2Vec as a Neural Network

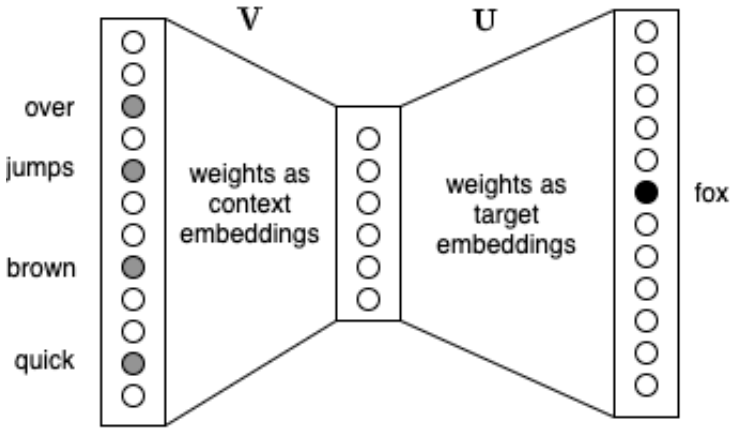
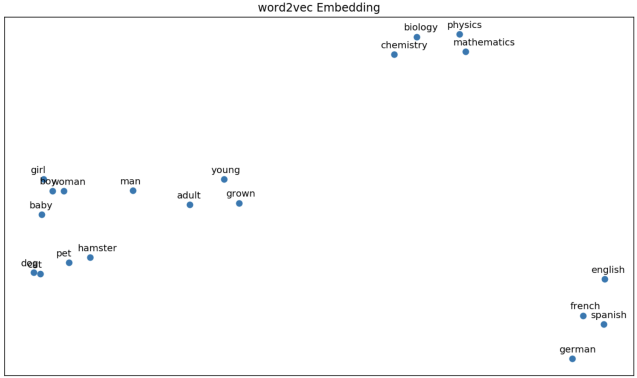


Figure adapted from machinelearningcoban.com/tabml_book/ch_embedding/word2vec.html

Visualizing Words in 2D



Words represented in a 2D space after dimensionality reduction

Figure adapted from jramkiss.github.io/2019/08/21/word-embeddings/

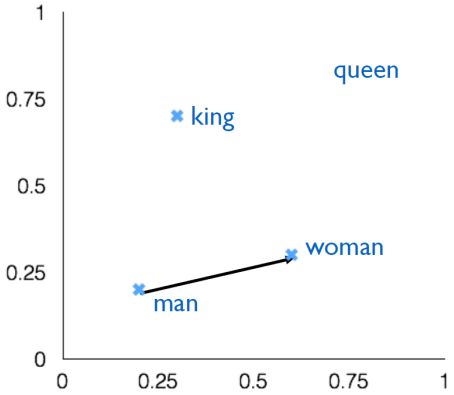
Word Analogy: Vector Arithmetic in Word2Vec

- Word2Vec embeddings can solve analogy tasks by performing vector arithmetic.
- The analogy task takes the form:

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

- The analogy is solved by finding the word vector closest to $\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman}$.

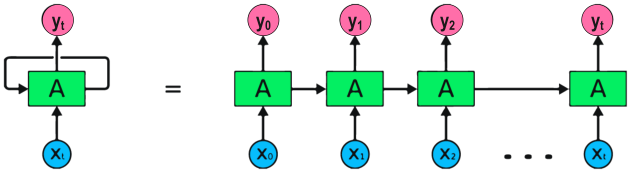
Word Analogy Example



Word analogy example

Recurrent Neural Networks

- To process each word, we need to remember the previous words.
- How do we create this memory? By maintaining a **hidden state**.
- Each hidden state captures information about:
  - Current word
  - All previous words in the sequence



where h_t serves as contextualized representation, containing memory of previous words.

Figure adapted from Abid Ali Awan. What are Recurrent Neural Networks (RNN), DataCamp

RNN's Hidden State Update

- Process sequences by maintaining a hidden state.
- Update state sequentially: $h_t = f_W(h_{t-1}, x_t)$
- This recurrence formula is applied at each time step to process a sequence of vectors x.
- The same function and the same set of parameters are used for each word.

$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W

Contextualized Word Embeddings

Compute contextual vector:

$$\mathbf{c_k} = f(w_k \mid w_1, w_2, \ldots, w_n) \in \mathbb{R}^d$$

Examples:

$f(\textit{light} \mid \text{Please turn off the } \overbrace{\textit{light}}^{} \text{ .})$

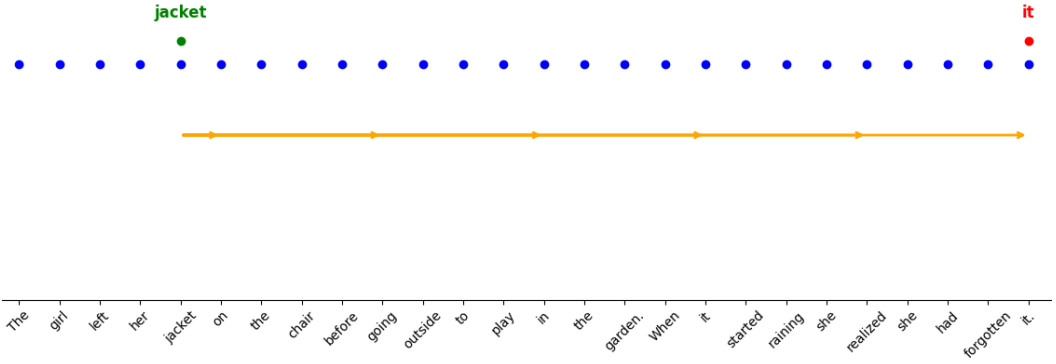
$\neq$

$f(\textit{light} \mid \text{This box is very } \underbrace{\textit{light}}_{} \text{ to carry.})$

How do we implement the context function f?

Challenges of RNNs

RNNs face challenges in retaining information over long sequences, such as when pronouns refer to distant words in the sequence.



So, what solution do we have?

Why Attention?

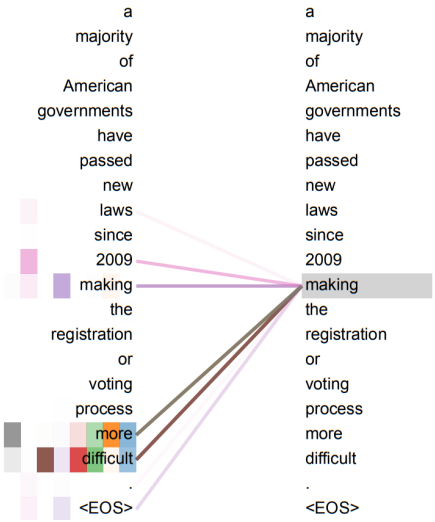
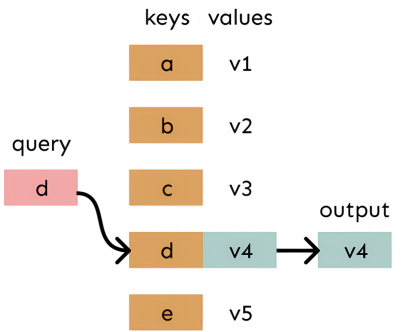
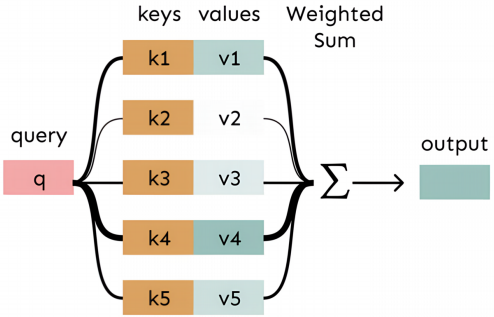


Figure adapted from Ashish Vaswani et al. Attention is All You Need paper

Attention as a Soft, Averaging Lookup Table



In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.



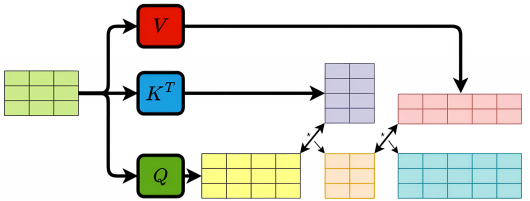
In the attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied and summed by the weights.

Figure adapted from Chris Manning, Lecture Slides, Stanford University

Self-Attention vs Cross-Attention

Self-Attention

- Queries, Keys, and Values are derived from the same sequence.
- Each position attends to all other positions in the sequence.



Cross-Attention

- Queries come from one sequence.
- Keys and Values come from another sequence.

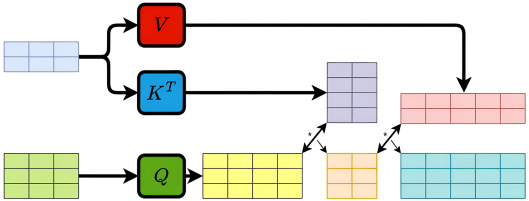
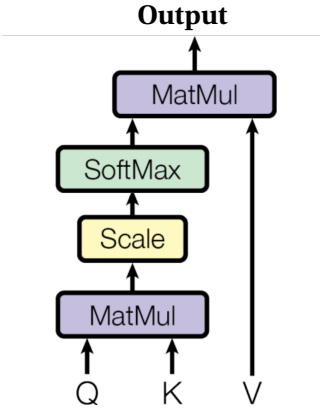


Figure adapted from Soran Ghaderi, Transformers-in-action-attention-is-all-you-need, TowardsDataScience blog

Scaled Dot-Product Attention



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

Figure adapted from Ashish Vaswani et al. Attention is All You Need paper

Multi-Head Attention

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

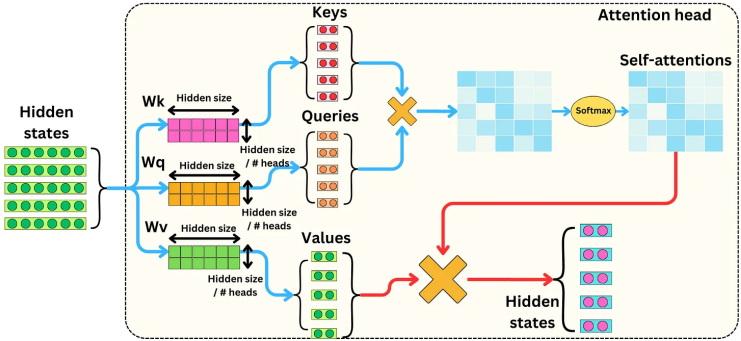


Figure adapted from Damien Benveniste, 'The Multi-head Attention Mechanism Explained,' The AIEdge Newsletter

Multi-Head Attention

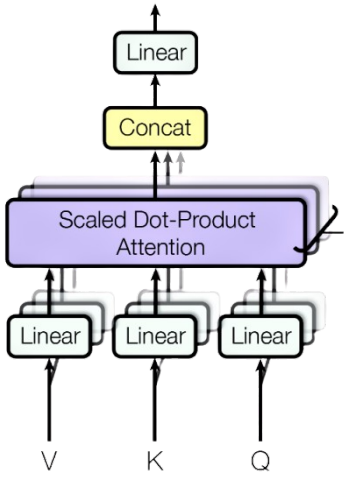
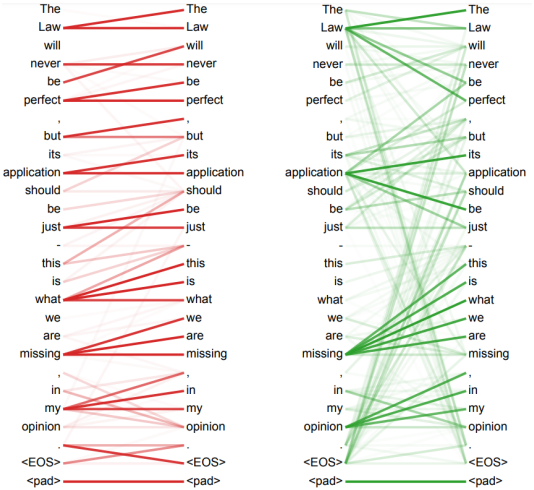


Figure adapted from Ashish Vaswani et al. Attention is All You Need paper | Kucedra : The 7 headed dragon, MythLok

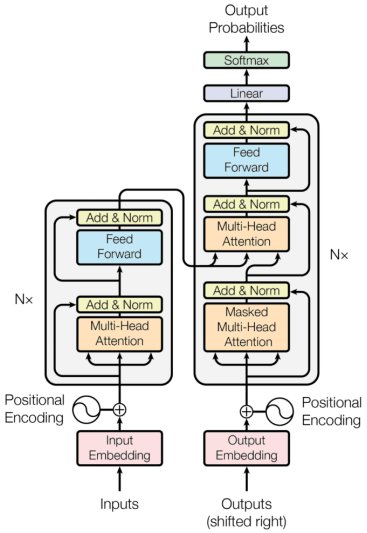
Multi-Head Attention



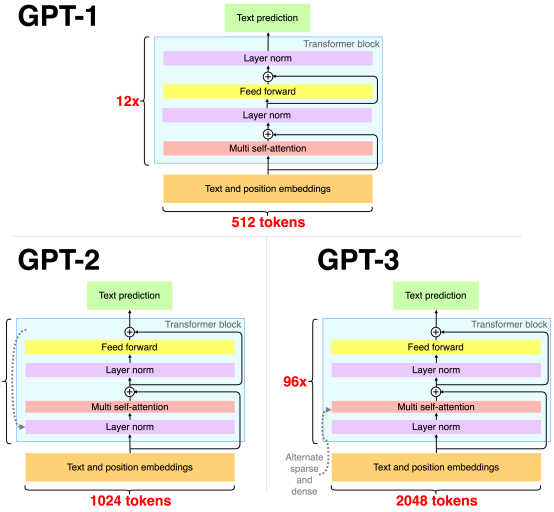
- Different attention heads seem to focus on different parts of the sentence.
- We show two examples from different heads.
- Each head appears to have learned a different task.

Figure adapted from Ashish Vaswani et al. Attention is All You Need paper

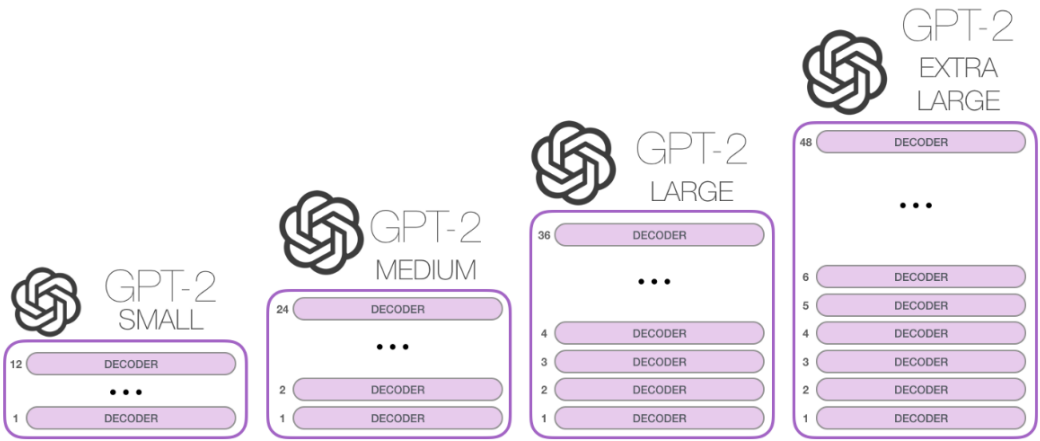
Transformers: A General Architecture



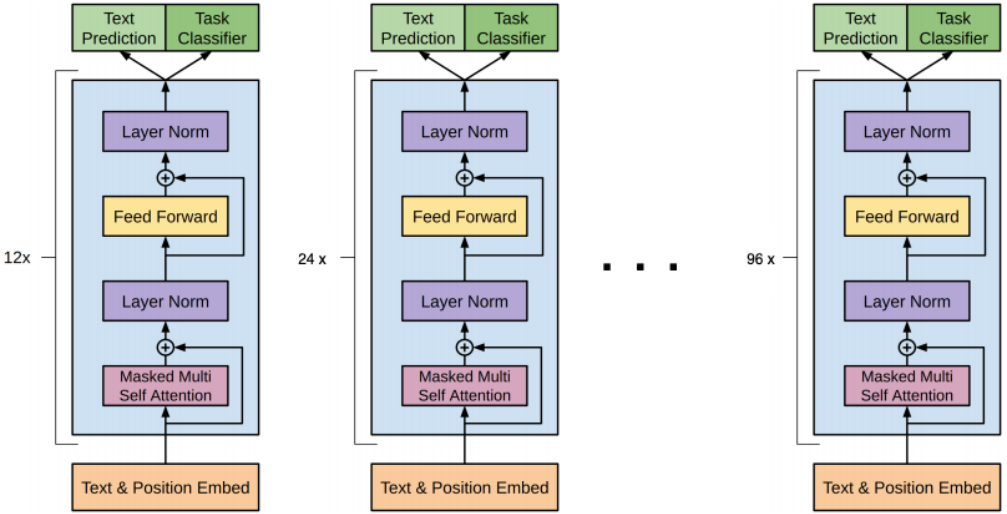
GPTs


GPT-1
Text prediction
Transformer block
Layer norm
Feed forward
Layer norm
Multi self-attention
12x
Text and position embeddings
512 tokens
GPT-2
Text prediction
Transformer block
Feed forward
Layer norm
Multi self-attention
Layer norm
48x
Text and position embeddings
1024 tokens
GPT-3
Text prediction
Transformer block
Feed forward
Layer norm
Multi self-attention
Layer norm
96x
Alternate sparse and dense
Text and position embeddings
2048 tokens

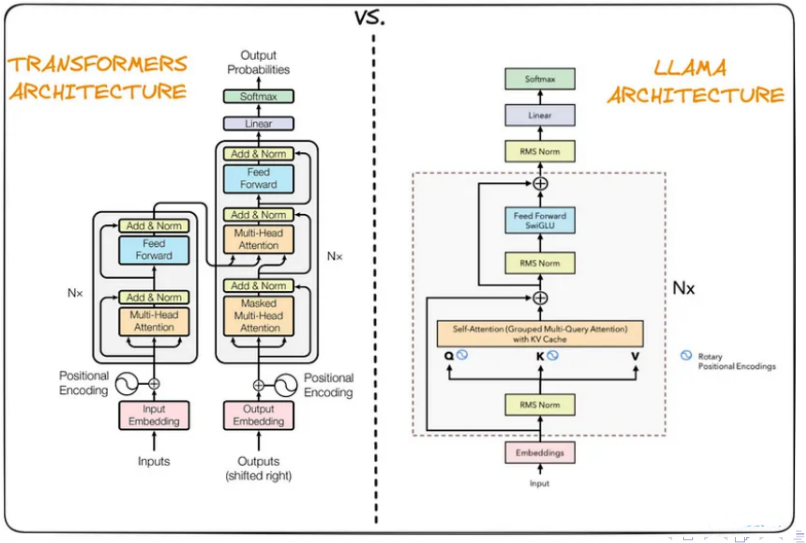
GPT-2


GPT-2
SMALL
12
DECODER
1
DECODER
Model Dimensionality: 768
GPT-2
MEDIUM
24
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1024
GPT-2
LARGE
36
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1280
GPT-2
EXTRA
LARGE
48
DECODER
6
DECODER
5
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1600

GPT-3


Text Prediction
Task Classifier
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
Text & Position Embed
12x
24 x
96 x

Llama 2 Architecture



Pre-training and adaptation

Figure 1: Overview of Model Training: Pre-training on large, unlabeled data builds foundational knowledge, while fine-tuning on smaller, labeled datasets adapts the model for specific tasks.

Adapters

- **Adapters:** New modules inserted between layers of a pre-trained model, with the original model weights fixed.
- **Training Efficiency:** Only adapter modules are tuned, initialized to ensure their output resembles that of the original model.