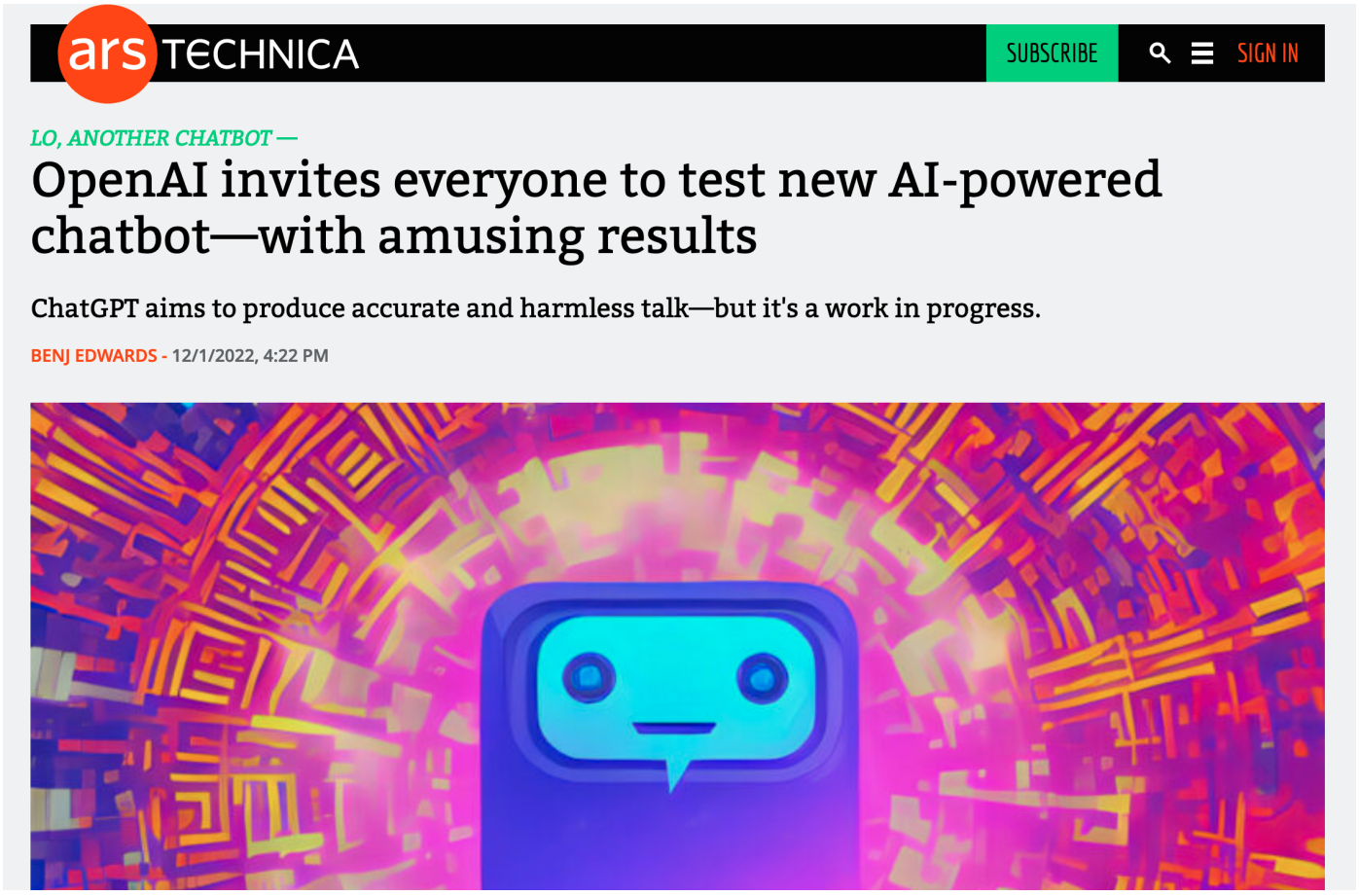
COMPSCI 589
Lecture 9: Transformers and LLMs

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

ChatGPT - 12/1/2022



https://arstechnica.com/information-technology/2022/12/openai-invites-everyone-to-test-new-ai-powered-chatbot-with-a

ChatGPT - 10/01/2024

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature › news › article

NEWS | 01 October 2024

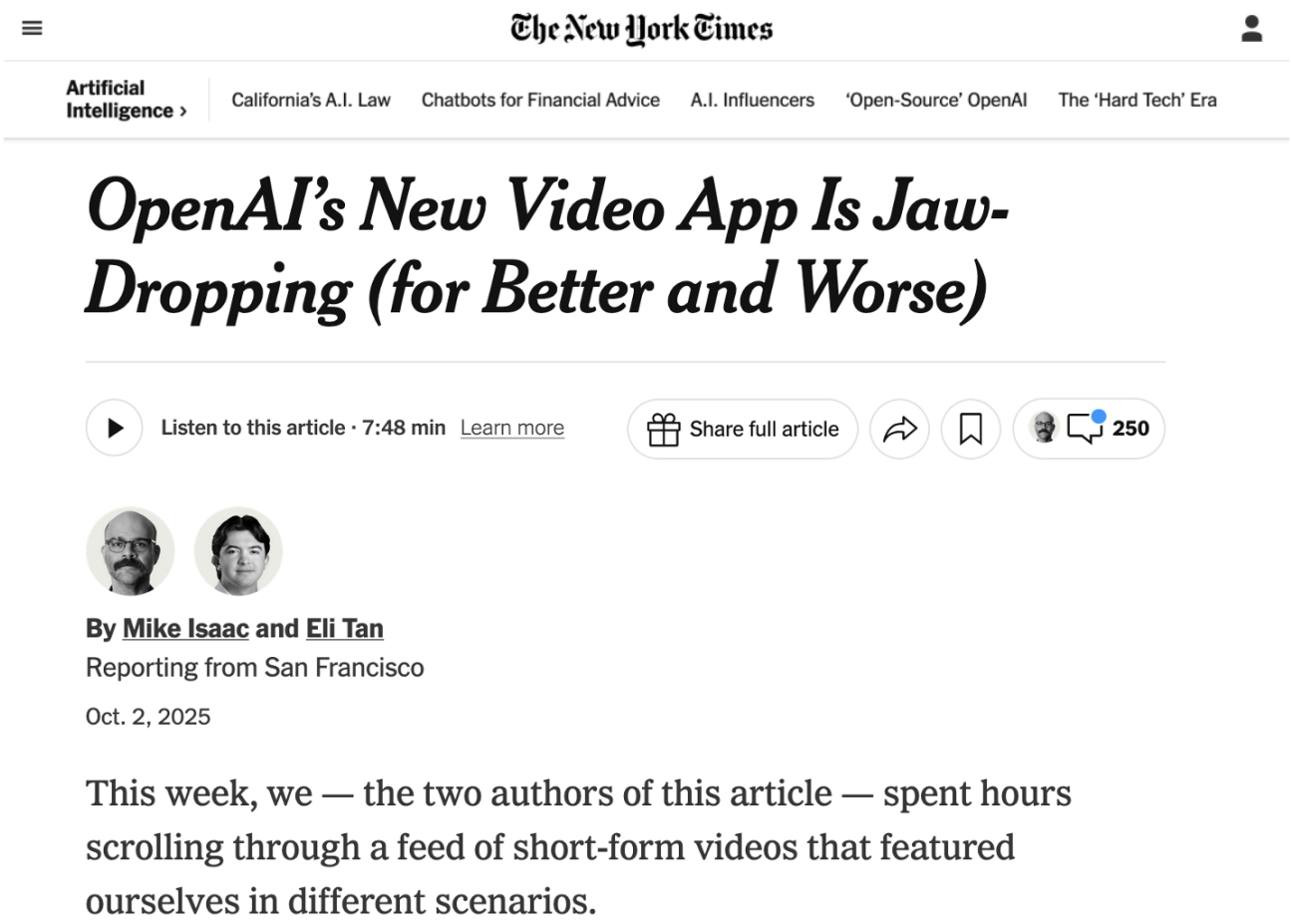
‘In awe’: scientists impressed by latest ChatGPT model o1

The chatbot excels at science, beating PhD scholars on a hard science test. But it might ‘hallucinate’ more than its predecessors.

By Nicola Jones

https://www.nature.com/articles/d41586-024-03169-9

Sora 2 - 10/2/2025



The New York Times

Artificial Intelligence › | California's A.I. Law | Chatbots for Financial Advice | A.I. Influencers | 'Open-Source' OpenAI | The 'Hard Tech' Era

OpenAI's New Video App Is Jaw-Dropping (for Better and Worse)

Listen to this article · 7:48 min Learn more | Share full article | 250

By Mike Isaac and Eli Tan
Reporting from San Francisco

Oct. 2, 2025

This week, we — the two authors of this article — spent hours scrolling through a feed of short-form videos that featured ourselves in different scenarios.

https://www.nytimes.com/2025/10/02/technology/openai-sora-video-app.html

Transformers

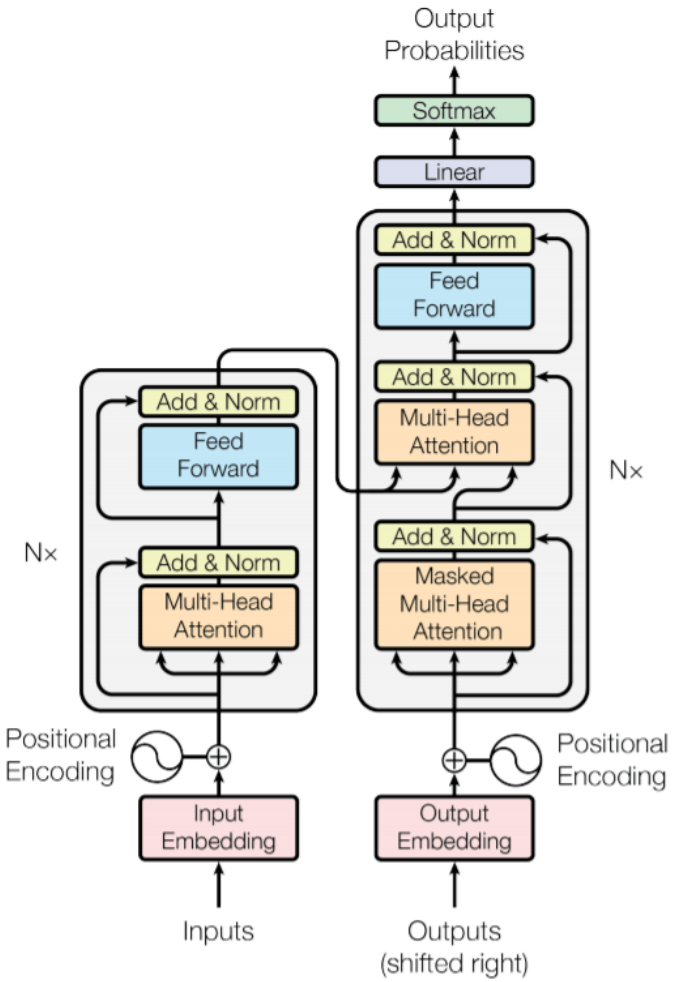


Image Credit: Ashish Vaswani et al.

Word Representation

- To model text data with neural networks, we need to convert words into vector representations. One approach is to represent each word using a “one-hot" encoding.
- If the vocabulary contains V words, we represent each word using a V-long vector where the v^{th} word in the vocabulary has a 1 in position v and all of the other positions are 0.
- For example: a = [1,0,...,0], able = [0,1,...,0], zoo = [0,0,...,1]
- In this representation, all words are at the same distance from each other in the representation space.
- Dissimilar words are not grouped together, but neither are similar words.

Learned Word Embeddings: Word2Vec

- A better approach is to learn word embeddings.
- The Word2Vec model learn word embeddings using multi-class logistic regression to predict the center word in a context window of five words.
- As input, we use the indices of the context words x_1, x_2, x_3, x_4.
- As output, we predict a probability distribution over the index of the central word, y.

Learned Word Embeddings: Word2Vec

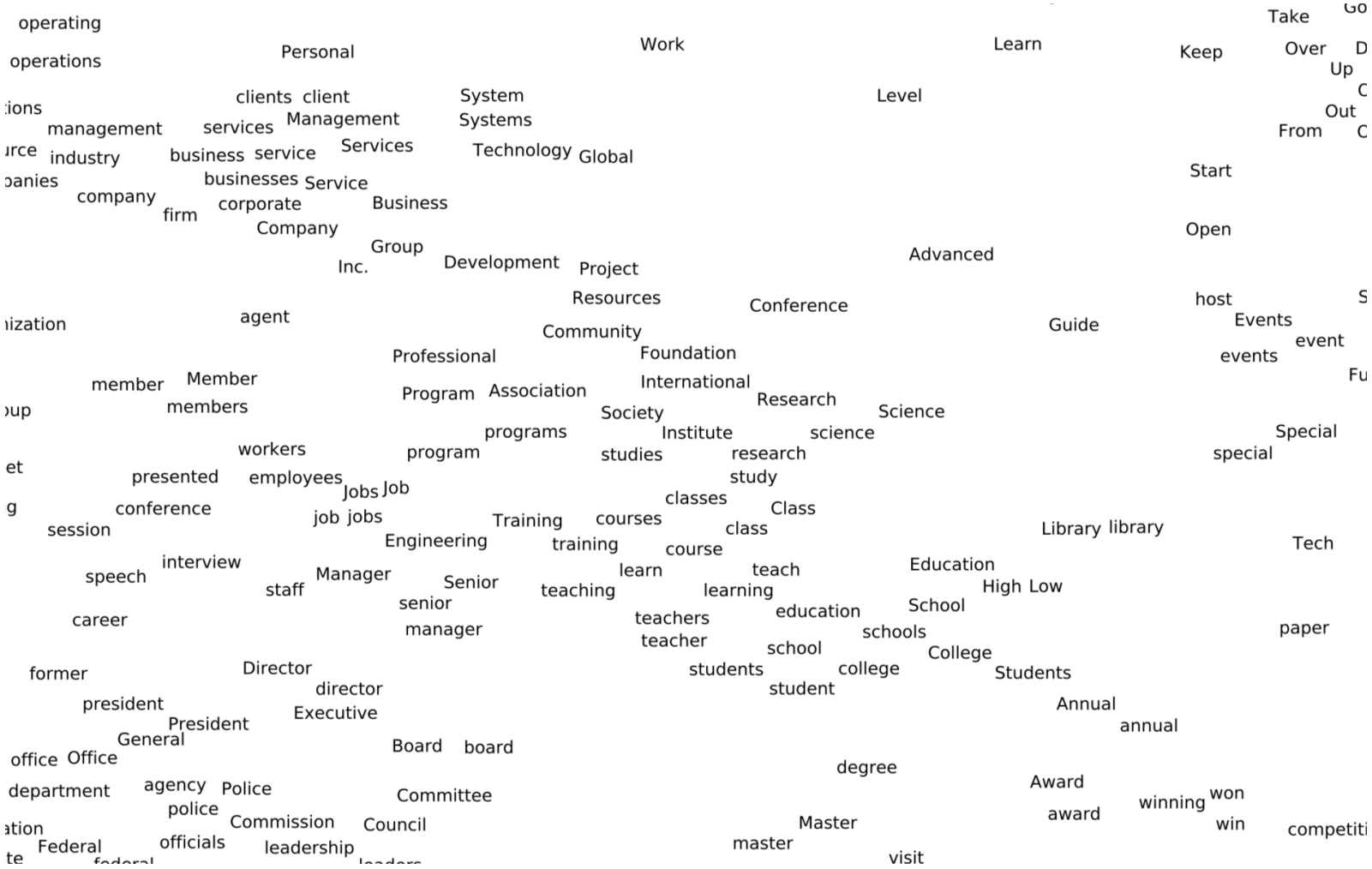
- The prediction function is a simple, one-hidden layer neural network with linear hidden activation and softmax output layer.

$$P(Y = y|x_1, ..., x_4) = \frac{\exp(\mathbf{z}^T \mathbf{w}_y^{out})}{\sum_{y'=1}^{V} \exp(\mathbf{z}^T \mathbf{w}_{y'}^{out})}$$

$$\mathbf{z} = \frac{1}{4} \sum_{j=1}^{4} \sum_{v=1}^{V} \mathbb{I}(x_j = v) \mathbf{w}_v^{embd}$$

- The model parameters $\mathbf{w}_v^{embd} \in \mathbb{R}^K$ are the embedding vectors of the words in the vocabulary.

Learned Word Embeddings: Word2Vec



Positional Encodings

- A word embedding is the same no matter where the word occurs in a sentence.
- To represent the position of a word in a sentence, we use a second embedding called a "positional encoding."
- A simple way to provide positional information is to use a one-hot encoding $\mathbf{p}_i$ of the position of each word in the sentence.
- This one-hot vector can be concatenated with the original word embedding vector $\tilde{\mathbf{x}}_i = [\mathbf{x}_i; \mathbf{p}_i]$ to represent the meaning of the word and the position of the word.

Positional Encodings

- The original transformer paper proposed a different approach.
- Instead of concatenating the position information with the word embedding information, the positional encoding vector is constrained to have the same length K as the word embedding and the two are added together: $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$.
- To enable this, the positional encoding can not be a one-hot encoding.
- In the original transformers paper, the positional encoding is a fixed mapping from sequence position to K-dimensional embedding space. Later work (e.g., GPT-2) used learned positional encodings.

Attention

- The fundamental building block of transformers is a computation referred to as *attention*.
- Given a sequence M input embedding vectors $\mathbf{x}_i$ of length K, the attention computation outputs a new representation of the M inputs as vectors $\mathbf{h}_i$ of length K.
- The self-attention computation allows for arbitrary mixing of information across the input embeddings regardless of the sequence length.

Computing Self-Attention

- The self-attention computation first transforms each input vector $\mathbf{x}_i$ into three different embeddings referred to as the key $\mathbf{k}_i$, the query $\mathbf{q}_i$ and the value $\mathbf{v}_i$, each of length K.
- All the transformations are linear. The parameters W^k, W^q and W^v are all $K \times K$ matrices.

$$\mathbf{k}_i = W^k \cdot \mathbf{x}_i$$
$$\mathbf{q}_i = W^q \cdot \mathbf{x}_i$$
$$\mathbf{v}_i = W^v \cdot \mathbf{x}_i$$

Computing Self-Attention

- The next step computes an attention weight a_{ij} for each pair of words (i, j).
- a_{ij} indicates how much word i will pay attention to word j when computing the output representation $\mathbf{h}_i$.
- The self-attention weights for word i are the softmax transform of the similarity between the query $\mathbf{q}_i$ for word i and the keys $\mathbf{k}_j$ for each word j in the sequence, with scaling applied:

$$a_{ij} = \frac{\exp(\frac{1}{\sqrt{k}}\mathbf{q}_i^T\mathbf{k}_j)}{\sum_{j'=1}^{T}\exp(\frac{1}{\sqrt{k}}\mathbf{q}_i^T\mathbf{k}_{j'})}$$

Computing Masked Self-Attention

- In order learn to predict next words in a sequence, transformers use masked self-attention instead of full self attention.
- Essentially, each sequence position can attend to itself and earlier sequence positions, but not later sequence positions.
- This requires a minor modification of the full self-attention computation that forces the attention values a_{ij} to normalize over the first i sequence positions. We set $a_{ij} = 0$ for $j > i$:

$$a_{ij} = \frac{\mathbb{I}[j \leq i] \exp(\frac{1}{\sqrt{k}} \mathbf{q}_i^T \mathbf{k}_j)}{\sum_{j'=1}^{T} \mathbb{I}[j \leq i] \exp(\frac{1}{\sqrt{k}} \mathbf{q}_i^T \mathbf{k}_{j'})}$$

Computing Output Embeddings

- The final output representation $\mathbf{h}_i$ for each object i is obtained as an attention-weighted linear combination of the values $\mathbf{v}_j$ of all objects (using full or masekd self-attention):

$$\mathbf{h}_i = \sum_{j=1}^{K} a_{ij} \mathbf{v}_j$$

- We can use this computation to define a self attention block that provides a mapping from a $M \times K$ representation of the M words $\mathbf{x}$ to a new $M \times K$ representation $\mathbf{h}$:

$$\mathbf{h} = \text{SelfAttention}_\theta(\mathbf{x})$$

Computing Self-Attention

- The parameters θ of the block self-attention$_\theta(\mathbf{x})$ are the three weight matrices $\theta = [W_k, W_q, W_v]$. The total parameter count is thus $3DK$.
- Note that similar to an RNN and CNN and unlike an MLP, the number of parameters used in the self-attention block is independent of the number of sequence length. It does depend on the length K of the individual input vectors.

Multi-Head Attention

- A single attention computation is often referred to as an "attention head."
- We can make models more complex by learning L different attention heads with different parameters θ_l.
- The multi-head attention computation can also produce a final representation of vectors of length k using an additional combination of the outputs of the multiple attention heads:

$$\mathbf{h}'_l = \text{SelfAttention}_{\theta_l}(\mathbf{x})$$

$$\mathbf{h}_i = \sum_{l=1}^{L} W_l^c \mathbf{h}'_{li}$$

Multi-Head Attention

- We can use the multi-head attention computation to define a multi-head attention block that also provides a mapping from a $M \times K$ representation of M objects $\mathbf{x}$ to a new $M \times K$ representation $\mathbf{h}$:

$$\mathbf{h} = \text{MultiheadAttention}_{\phi}(\mathbf{x})$$

- The parameters ϕ include the parameters θ_l for each of the L self-attention blocks, as well as the multi-head attention combination weights W^c.

Transformer Blocks

- Apart from the softmax function used to normalize the attention weights, the attention computation is a linear function of the input embeddings.
- To give the model more representational power, an MLP is typically applied to the output embedding of each sequence position to make the encoding non-linear.

Transformer Blocks

- The full transformer block used in the original paper applies multi-head attention, layer norm, a one-hidden layer relu MLP, and residual connections:

$$
\begin{aligned}
\mathbf{h} &= \text{MultiheadAttention}_{\phi}(\mathbf{x}) \\
\mathbf{u}_i &= \text{LayerNorm}(\mathbf{x}_i + \mathbf{h}_i, \gamma_1, \beta_1) \\
\mathbf{z}'_i &= W_2 \text{ relu}(W_1 \mathbf{u}_i) \\
\mathbf{z}_i &= \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}'_i, \gamma_2, \beta_2)
\end{aligned}
$$

- Other architectures like the GPT family use different permutations of these components.

Aside: Layer Norm

- The LayerNorm block performs a re-scaling of the values within each vector:

$$\text{LayerNorm}(\mathbf{z}, \gamma, \beta) = \beta + \gamma \frac{(\mathbf{z} - \mu(\mathbf{z}))}{\sigma(\mathbf{z})}$$

$$\mu(\mathbf{z}) = \frac{1}{K} \sum_{k=1}^{K} \mathbf{z}_k$$

$$\sigma(\mathbf{z}) = \left(\frac{1}{K} \sum_{k=1}^{K} (\mathbf{z}_k - \mu(\mathbf{z}))^2 \right)^{1/2}$$

Transformer Blocks

- We can use the transformer block computation to define a new transformation computation:

$$\mathbf{z} = \text{TransformerBlock}_{\omega}(\mathbf{x})$$

- Here the parameters ω include the parameters ϕ of the included multi-head attention computation, and the parameters γ_1, γ_2, β_1, β_2, W_1 and W_2 used in the rest of the transformer block computation.

Transformer Models

- A transformer model is simply a stack of R transformer blocks each with their own parameters ω_r.
- Each block in the stack takes its input from the previous block, except for the first block, which takes its input from $\mathbf{x}$.

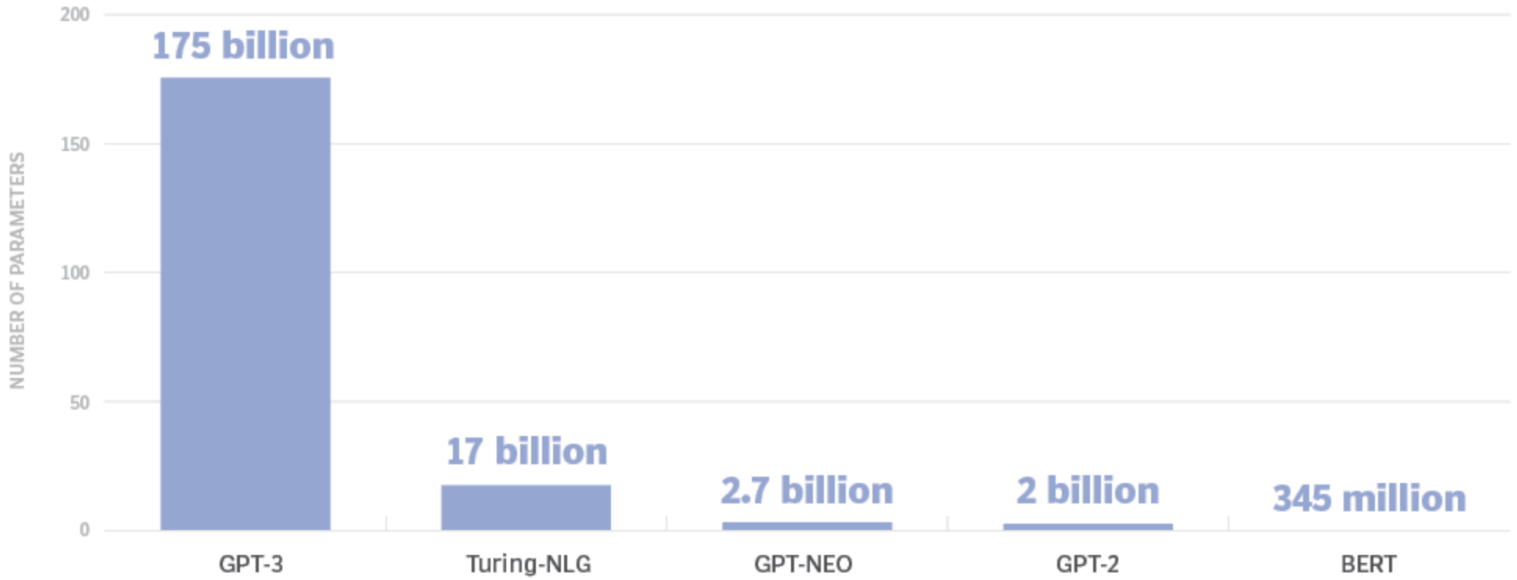
$$\begin{aligned}
\mathbf{z}^1 &= \text{TransformerBlock}_{\omega_1}(\mathbf{x}) \\
\mathbf{z}^2 &= \text{TransformerBlock}_{\omega_2}(\mathbf{z}^1) \\
&\vdots \\
\mathbf{z}^R &= \text{TransformerBlock}_{\omega_R}(\mathbf{z}^{R-1})
\end{aligned}$$

- The blocks can use full self-attention or masked self-attention depending on the task.

Generative Pre-Training

- Transformer models are typically pre-trained on the language modeling task.
- This task is almost the same as the Word2Vec training task except that instead of predicting a word in the middle of a sequence, we predict the next word.
- The GPT family of models do this using a stack of masked self-attention transformer blocks.
- To predict the next word $i + 1$, the embedding of the last word in the top block $\mathbf{z}_i^R$ is used as the feature vector in a multi-class logistic regression classifier that produces a probability distribution over words in the vocabulary.
- The model is trained to minimize the negative average log likelihood of the predicted next word distribution.

Text Transformer Model Parameters



GPT4 is reported to have 1.5 trillion parameters. Note: the human brain has about 85 billion neurons and 100 trillion synapses.

Post-Training and RLHF