COMPSCI 689
Lecture 23: Transformers

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).

ChatGPT - 12/1/2022

ars TECHNICA SUBSCRIBE SIGN IN

LO, ANOTHER CHATBOT —

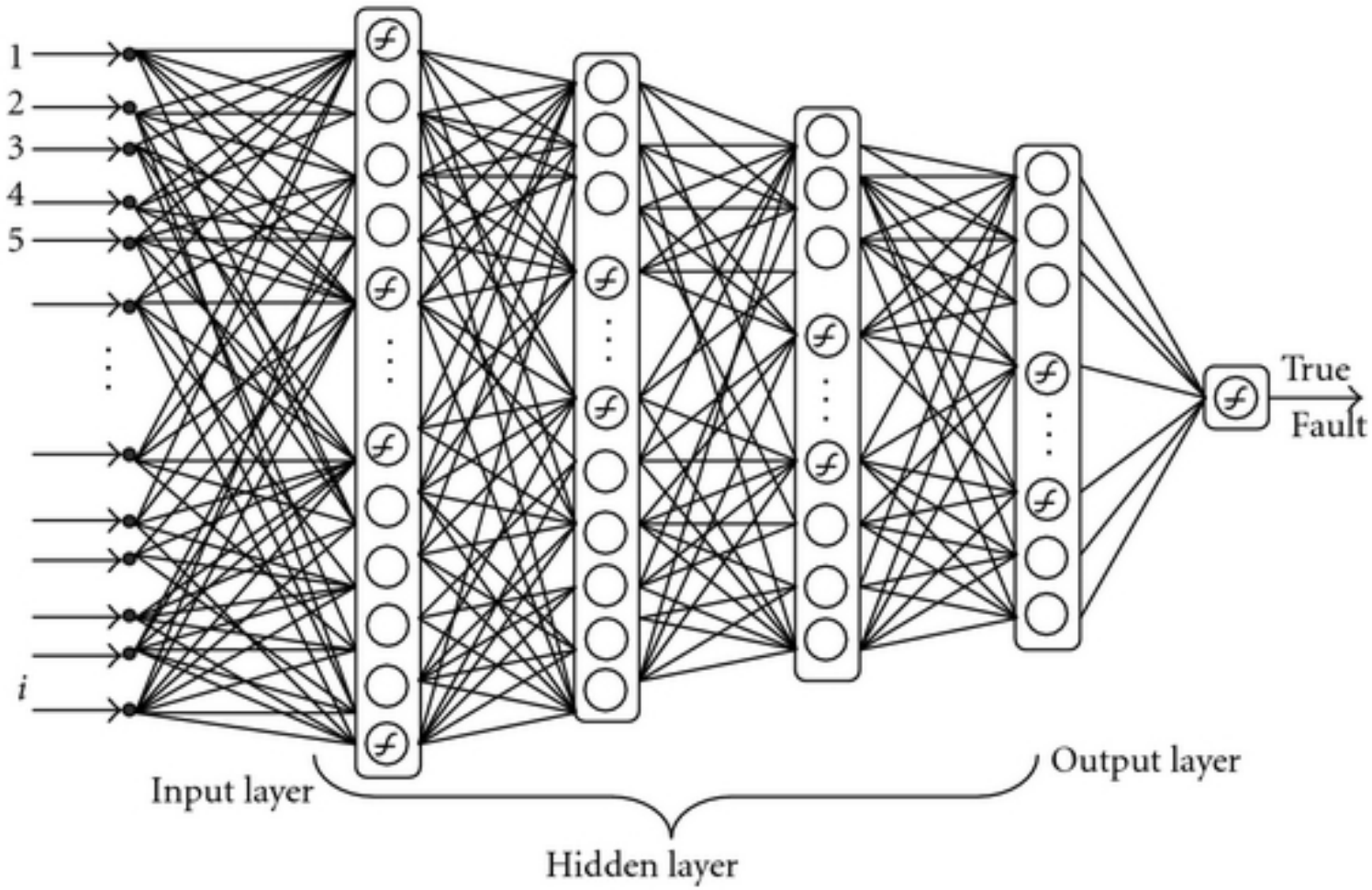
OpenAI invites everyone to test new AI-powered chatbot—with amusing results

ChatGPT aims to produce accurate and harmless talk—but it's a work in progress.

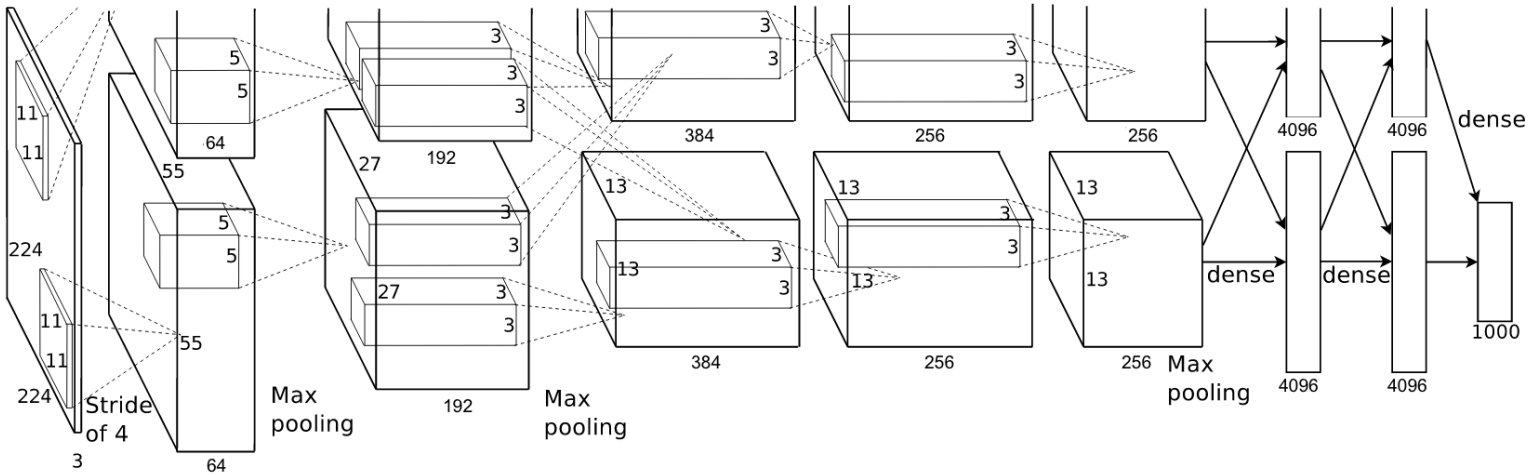
BENJ EDWARDS - 12/1/2022, 4:22 PM

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

Multi-Layer Perceptrons



Convolutional Models




```
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(64, 192, kernel_size=5, padding=2),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(192, 384, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
)
```

```
self.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, num_classes),
)
```

Recurrent Neural Networks

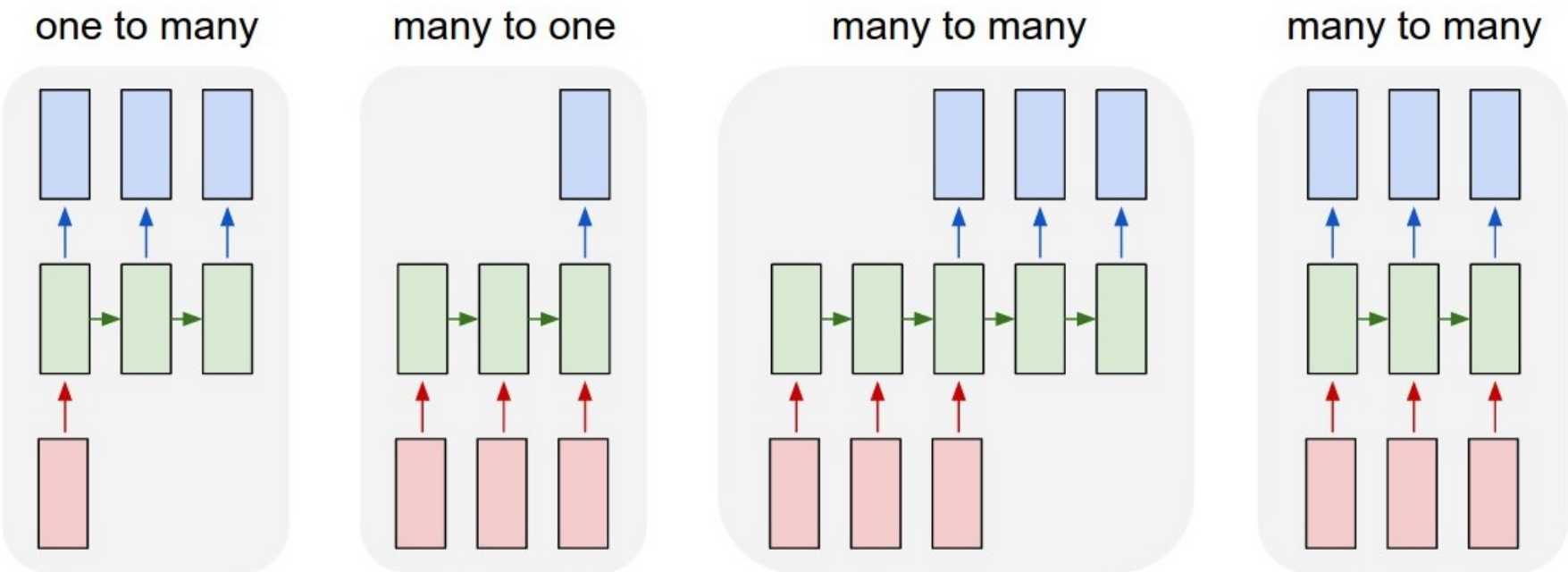


Image Credit: Andrej Karpathy

Bi-Directional Recurrent Networks

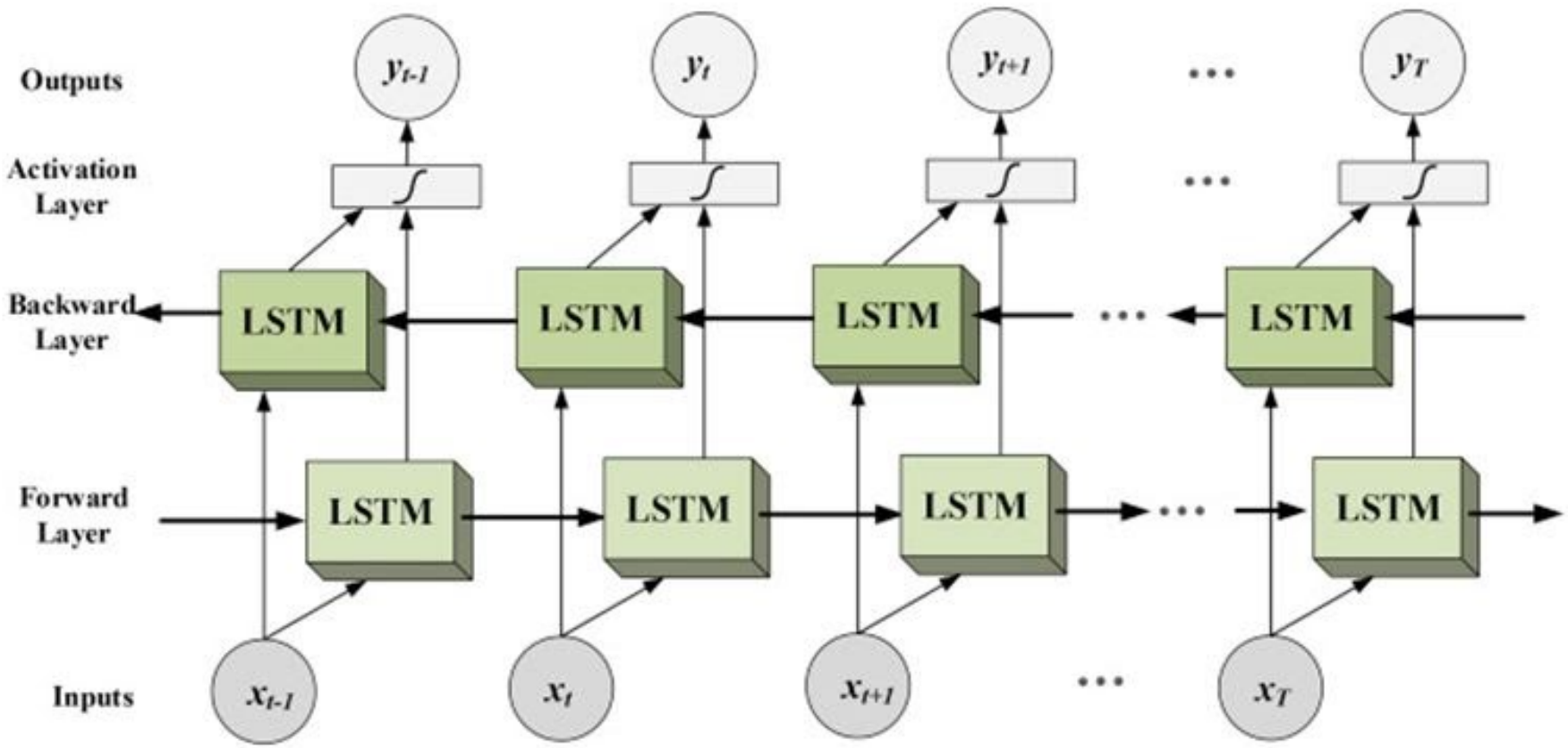


Image Credit: Yugesh Verma

Transformers

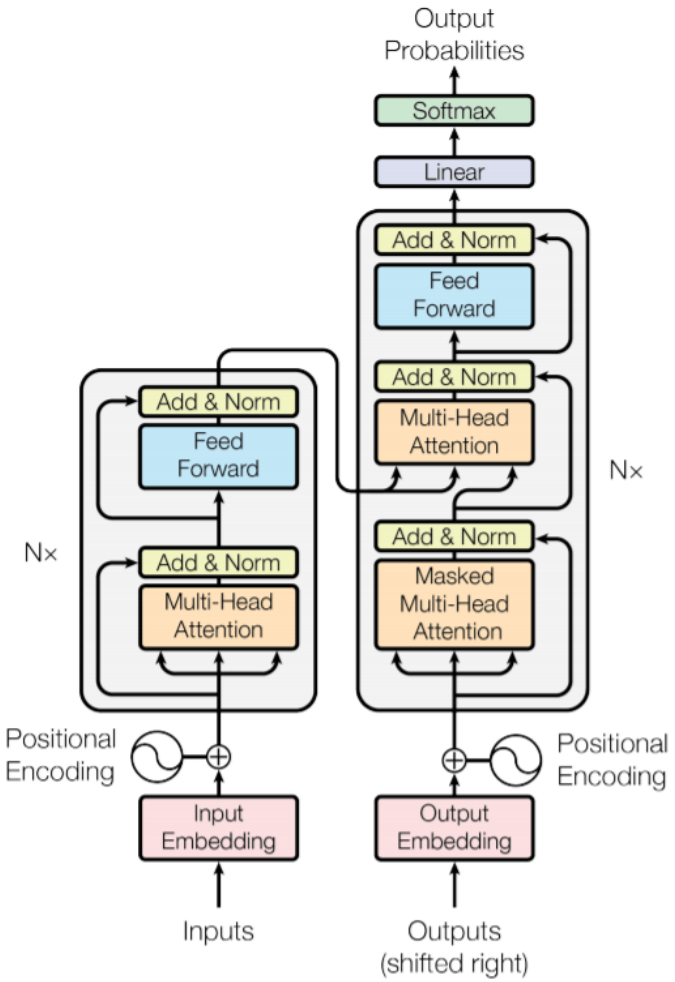


Image Credit: Ashish Vaswani et al.

Attention

- The fundamental building block of transformers is a computation referred to as *attention*.
- Given an input consisting of a collection of M objects represented as column vectors $\mathbf{x}_i$ of length D, the attention computation outputs a new representation of the M objects as column vectors $\mathbf{h}_i$ of length K.
- The attention computation allows for arbitrary mixing of information across the collections of objects regardless of the number of objects.

Computing 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 $K \times D$ 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 Attention

- The next step computes an attention weight a_{ij} for each pair of input objects (i, j).
- a_{ij} indicates how much object i will pay attention to object j when computing the output representation $\mathbf{h}_i$.
- The attention weights for object i are the softmax transform of the similarity between the query $\mathbf{q}_i$ for object i and the keys $\mathbf{k}_j$ for each object j in the collection, 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 Attention

- 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:

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

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

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

Computing Self-Attention

- The parameters θ 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 objects. It does depend on the length D of the individual input vectors.

Multi-Head Attention

- A single self-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 $M \times D$ representation of M objects $\mathbf{x}$ to an $M \times K$ 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.

Transformers

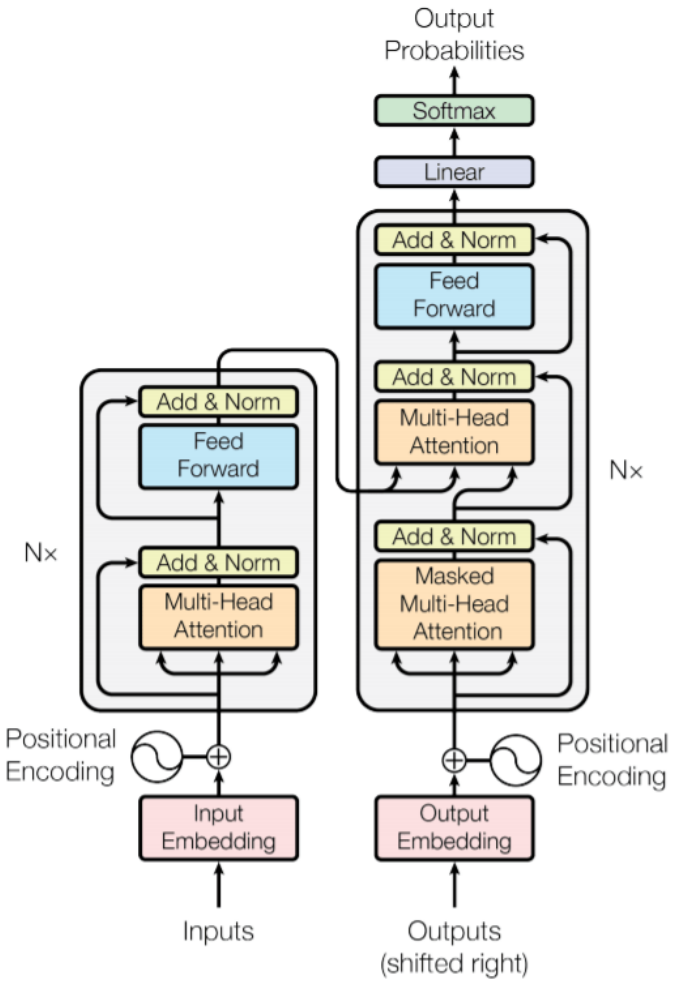


Image Credit: Ashish Vaswani et al.

Transformer Blocks

- A transformer block is a multi-head attention-based module that also performs a mapping from a collection of M inputs in D dimensions to a collection of M objects in K dimensions.
- It starts with the multi-head attention computation and then performs several additional layers of non-linear transformations that act independently on the representations output by the multi-head attention process:

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

Aside: Layer Norm

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

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

$$\mu(\mathbf{z}) = \frac{1}{K} \sum_{i=1}^{K} \mathbf{z}_i$$

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

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.

Transformers

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

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

Transformer Applications

- Note that since the transformer operates on representations of a collection of objects, it is a highly general model.
- The original applications of transformers were in natural language processing. They were then extended to models for multivariate time series and images.

Transformers For Sequences

- For sequential data such as text and time series, the first step in applying a transformer is to form a multi-dimensional representation of each sequence element.
- For multivariate time series, the data at each time point are already a vector of some length D.
- For text, it is common to embed the data at the word level. This can be done using fixed one-hot encoding, a fixed previosuly learned embedding, or a learnable embedding.

Representing Structure

- Since the transformer treats it's inputs as a collection of objects, it is actually invariant to the sequential ordering of the inputs.
- As a result, transformers need to be provided with additional data about the structure of the collection of objects.
- For time series and text, the model must be provided with a representation of the sequential ordering of the objects. This is generally referred to as a "positional encoding."

Positional Encodings

- A simple way to provide a transformer block with positional information is to use a one-hot encoding $p(i)$ of the position of each input data vector $\mathbf{x}_i$ in the sequence.
- This one-hot vector can be concatenated with the original data vector $\tilde{\mathbf{x}}_i = [\mathbf{x}_i; p(i)]$. The transformer then operates using $\tilde{\mathbf{x}}_i$ as input.
- Note that when stacking transformer blocks, we may want to re-inject positional encoding information at the input to each block in the stack.

Positional Encodings

- In the original transformers paper, the proposed model uses a different approach that can be thought of as producing an embedding of positional information $p(i)$ of size D for each position i in the input sequence.
- Instead of concatenating the position information with the feature information, the original model adds the positional encoding vector to the data vectors yielding $\tilde{\mathbf{x}}_i = \mathbf{x}_i + p(i)$. The transformer then operates using $\tilde{\mathbf{x}}_i$ as input.
- Again, this can be repeated at the input to each transformer block.

Transformers

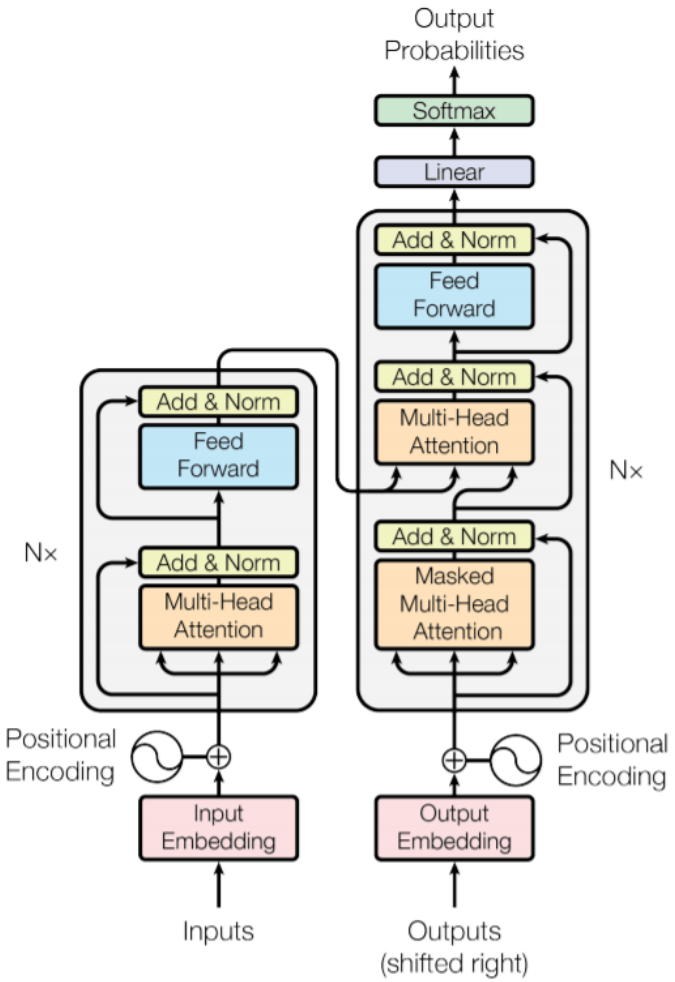


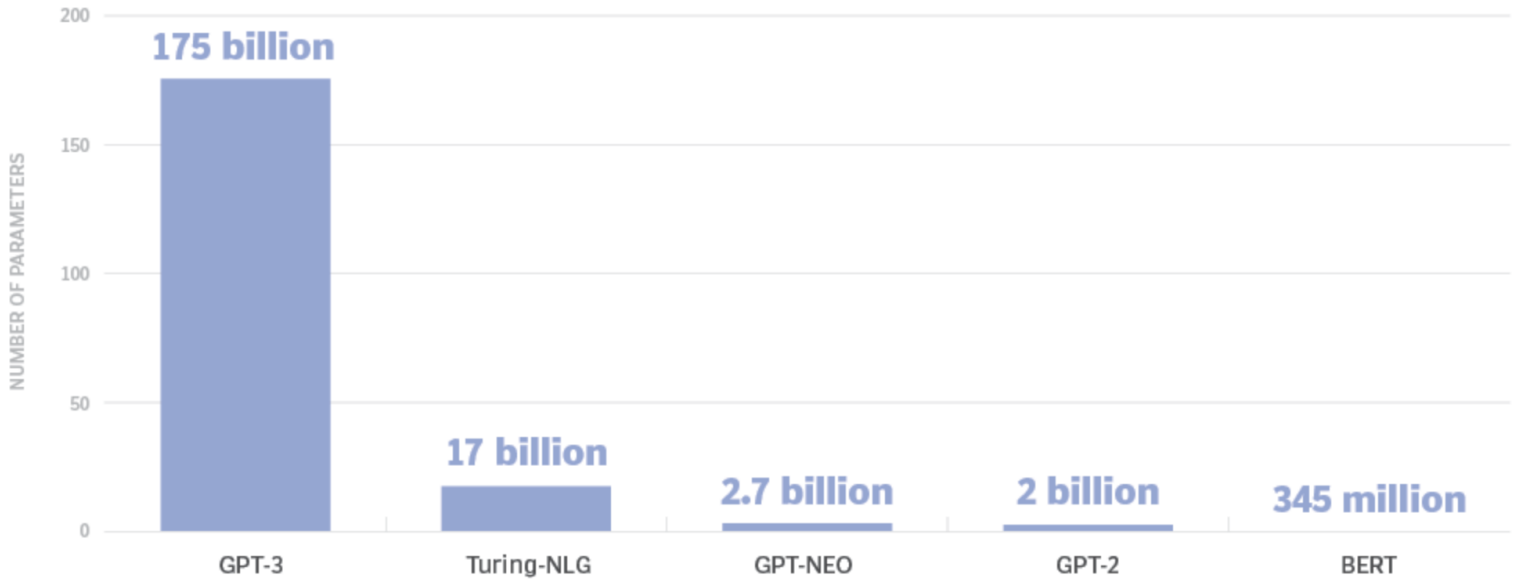
Image Credit: Ashish Vaswani et al.

Models for Text

- In the case of text data, current transformer models are trained as unsupervised models by learning to predict the next word in a sequence.
- The output of the final transformer block is a $M \times K$ matrix.
- We can select a given context window size C and use the final representations in this window to predict the probability of the next word in the sequence.

$$p(W_{M+1} = w | \mathbf{x}_1, ..., \mathbf{x}_M) = \frac{\exp\left(\sum_{i=1}^{C} \sum_{k=1}^{K} W^p_{wik} \cdot \mathbf{z}^R_{i+\delta,k}\right)}{\sum_{w'=1}^{V} \exp\left(\sum_{i=1}^{C} \sum_{k=1}^{K} W^p_{w'ik} \cdot \mathbf{z}^R_{i+\delta,k}\right)}$$

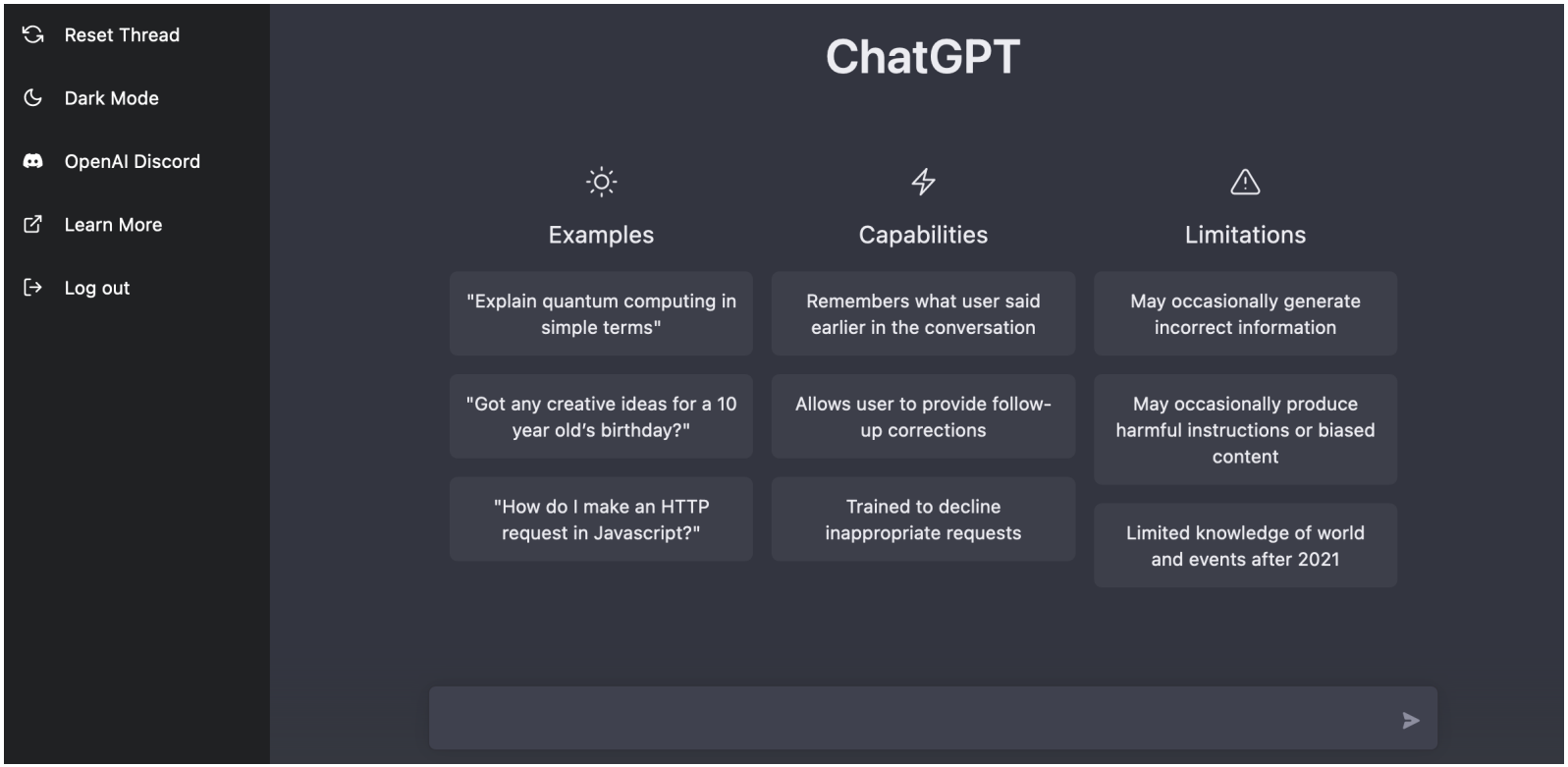
Text Transformer Model Parameters



GPT-3 Models Sizes

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Chat GPT-3



https://openai.com/blog/chatgpt/