Transformer

Initialization:
Input is a set (or sequence) of tokens from a finite dictionary.

Eg. Text: She works in the white house.

Map tokens to word vectors: \( x_i \in \mathbb{R}^d \)
These vectors are learned

Add positional embedding

\[
\mathbf{z} = [z_1, \ldots, z_n]
\]

\[
\mathbf{z} \leftarrow \mathbf{z} + \mathbf{p}
\]

\[
p_{i,j} = \sin \left( \frac{j}{n^{\frac{1}{2}}} \right)
\]

\[
p_{i,j} = \cos \left( \frac{j}{n^{\frac{1}{2}}} \right)
\]

The vector \( z_i \) distills the "meaning" of the \( i \)th word. Similar words have similar embeddings, e.g., "white" ≠ "blue".

Issue: The vector \( z_5 \) of "white" should change as it is followed by "house". Words modify each other's meaning.

(Self-)Attention: High level: Each vector becomes a weighted average of its nearest neighbors and itself.

\[
\begin{align*}
\mathbf{z}_i & \approx \sum_{j \neq i} \mathbf{w}_{ij} \mathbf{z}_j \\
\mathbf{w}_{ij} & = \frac{\exp(\text{sim}(\mathbf{q}_i, \mathbf{k}_j))}{\sum_{j' \neq i} \exp(\text{sim}(\mathbf{q}_i, \mathbf{k}_{j'}))}
\end{align*}
\]

Similarity measure between \( z_i \) and \( z_j \): \( \text{sim}(\mathbf{q}_i, \mathbf{k}_j) = \frac{q_i \cdot k_j}{\|q_i\| \cdot \|k_j\|} \)

Weight of \( z_j \) as neighbor for \( z_i \): \( w_{ij} = \frac{q_i \cdot k_j}{\sum_{j' \neq i} \exp(\text{sim}(\mathbf{q}_i, \mathbf{k}_{j'}))} \) softmax function

Self-attention:

\[
\mathbf{z}_i \leftarrow \sum_{j \neq i} w_{ij} \mathbf{z}_j
\]

Skip connection

\[
\mathbf{z} \leftarrow \mathbf{z} + \mathbf{V} \mathbf{\text{softmax}(K^T Q^T \mathbf{z})} + \mathbf{z}
\]

\( \mathbf{W} \in \mathbb{R}^{m \times d} \)
**Feed-Forward Neural Net:**

After the self-attention, each vector is processed by a FFN (MLP).

\[ z_i'' = \text{ReLU}(\text{MLP}(z_i')) + z_i' \]

**Transformer Layer:**

- **SA:** Self-Attention
- **LN:** Layer Normalization
- **FFN:** Feed Forward Neural Net (MLP)

**LLM:** Predict the next word:

I like to eat

- bananas 0.1
- cherries 0.05
- fruit 0.03
- cake 0.02

Output: K-class classification problem

Encoder turns context into word vectors.

Decoder predicts the next word. After a word is predicted it is inserted into the decoder as next input.

**Sampling:** The decoder outputs a probability vector. With \( p \) sampling samples words up to a probability mass of \( p \). E.g. \( p = 0.18 \) only sample [bananas, cherries, fruit].

**Emergent abilities:** Large Language Models learn to reason, do elementary math.