Embeddings and Self-Attention in Transformer Models

Embeddings: Turning Tokens into Vectors

What Are Embeddings?

Embeddings are dense vector representations of tokens — which can be words, subwords, or characters. Instead of representing words as one-hot vectors (which are sparse and uninformative), embeddings map each token to a continuous vector that captures its meaning and relationships to other tokens.

For example, the words “king” and “queen” might have similar embeddings because they share semantic features like royalty and gender.

How Embeddings Work

1. **Tokenization**: Text is broken into smaller units called tokens. These could be full words (“cat”), subwords (“##ing”), or even characters.
2. **Embedding Matrix**: Each token is mapped to a vector using a lookup table. This table is a matrix of size V × d, where:
    - V is the vocabulary size.
    - d is the embedding dimension.

    Formula: $\text{Embedding}(t_i) = E[t_i]$

    This means the embedding of token t_i is simply the row in matrix E corresponding to that token.

3. **Semantic Representation**: Words with similar meanings end up with similar vectors, allowing the model to generalize better across contexts.

Positional Encoding

Transformers process input in parallel, so they don’t inherently understand the order of words. To fix this, positional encodings are added to the embeddings to provide information about word position.

There are two types:

- **Sinusoidal Encoding**: Uses sine and cosine functions to encode position.
- **Learned Encoding**: Uses trainable vectors.

Formula (Sinusoidal): PE(pos, 2i) = sin(pos / 10000^(2i/d)) PE(pos, 2i+1) = cos(pos / 10000^(2i/d))

This ensures each position has a unique encoding that can be added to the token embedding.

Self-Attention: Understanding Context

What Is Self-Attention?

Self-attention allows each word in a sentence to consider other words when forming its representation. This is crucial for understanding context, especially in complex sentences.

For example, in “The bank raised interest rates,” the word “bank” could mean a financial institution or a riverbank. Self-attention helps the model decide based on surrounding words.

How Self-Attention Works

Each token is transformed into three vectors:

- **Query (Q)**: What this word wants to know.
- **Key (K)**: What this word offers.
- **Value (V)**: The actual information this word carries.

These are computed using learned weight matrices: $Q = X \times W^{\varphi}$ $K = X \times W^{k}$ $V = X \times W^{v}$

Where X is the input embedding matrix, and W^{φ}, W^{k}, W^{v} are trainable parameters.

Calculating Attention

To determine how much attention one word should pay to another, we compute the dot product between the query and key vectors, scale it, and apply softmax to get attention weights.

Formula: $\text{Attention}(Q, K, V) = \text{softmax}((Q \times K^{T}) / \sqrt{d_k}) \times V$

Explanation:

- $Q \times K^{T}$: Measures similarity between queries and keys.
- $\sqrt{d_k}$: Prevents large values that could destabilize softmax.
- Softmax: Converts scores into probabilities.
- Final output: Weighted sum of value vectors based on attention scores.

Multi-Head Attention and Transformer Layers

Multi-Head Attention

Instead of using a single attention mechanism, transformers use multiple attention “heads.” Each head learns to focus on different aspects of the sentence — grammar, meaning, relationships, etc.

Formula: $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h) \times W^{o}$

Each head performs its own attention calculation, and the results are concatenated and linearly transformed.

Transformer Layer Structure

Each transformer layer includes:

1. **Multi-Head Attention**
2. **Add & Normalize**: Combines input and output, then normalizes.
3. **Feedforward Network**: Applies further transformations.
4. **Add & Normalize Again**

The feedforward network typically consists of two linear layers with a ReLU activation in between.

Residual Connections: These help preserve information and stabilize training: Output = LayerNorm(x + SubLayer(x))

Applications and Interpretability

Applications in Generative AI

- **Text Generation**: GPT models use embeddings and self-attention to generate coherent text.
- **Translation**: Transformers outperform older models in translating languages.
- **Summarization**: They can extract key ideas from long documents.
- **Code Generation**: Models like Codex use token embeddings of programming languages.

Understanding the Model

- **Attention Maps**: Visual tools that show which words the model is focusing on.
- **Embedding Visualization**: Techniques like PCA or t-SNE can show how similar words cluster together.
- **Head Specialization**: Some attention heads focus on syntax, others on semantics or coreference.

Challenges

- Attention weights don’t always explain model decisions clearly.
- Embeddings can drift during fine-tuning.
- Positional encoding may struggle with very long sequences.