Embeddings and Self-Attention in AI and Generative AI

Introduction and High-Level Overview

Embeddings and self-attention are core building blocks of modern AI and generative AI systems. Embeddings convert discrete items such as words, subwords, image patches, or tokens into dense continuous vectors that capture semantic and structural relationships. Self-attention enables a model to compute contextualized representations by dynamically weighting interactions between all tokens in a sequence. Together these mechanisms let models represent meaning in vector space and compute context-sensitive transformations that power translation, summarization, code generation, multimodal understanding, and image generation.

Key benefits and intuition

- **Semantic geometry**: embeddings place related items near each other so similarity is measurable by vector distances.
- **Contextualization**: self-attention produces outputs where each token's vector reflects information aggregated from the whole input, not just local neighbors.
- **Parallelism and flexibility**: attention can be computed in parallel for all tokens and adapts to any token-pair relationships, enabling long-range dependencies and multimodal fusion.

Concrete example to anchor intuition

- Words "bank" in "river bank" vs "savings bank" have the same token but different meaning. An embedding alone is ambiguous; self-attention lets the model use surrounding tokens to shift the representation toward the river sense or the finance sense.

Embeddings Deep Dive

Definition and purpose

- **Embedding**: a dense vector representation x ∈ ℝ^d for a discrete symbol. The dimension d is chosen based on model capacity and downstream tasks. Embeddings compress syntactic and semantic properties into numeric form so neural networks can operate on them.

Types of embeddings

- **Word and subword embeddings**: learned lookup tables for tokens produced by tokenizers such as BPE or WordPiece.
- **Contextual embeddings**: produced by models with attention where the same token has different vectors depending on surrounding tokens.
- **Positional embeddings**: added to token embeddings to encode token order. Can be static sinusoidal functions or learned vectors.
- **Multimodal embeddings**: map non-text modalities (image patches, audio frames) to the same vector space so cross-modal attention is possible.

How embeddings are trained

- **Direct supervised learning**: embeddings are learned as part of a model trained end-to-end on a downstream task.

- **Self-supervised pretraining**: embeddings are trained within large models using objectives such as masked token prediction or autoregressive next-token prediction.
- **Metric learning**: embeddings learned with contrastive losses bring similar items closer and push dissimilar ones apart.

Concrete numeric example

- Suppose vocabulary includes tokens: “cat”, “dog”, “apple”, with embedding dimension d = 4. Example learned embeddings might be:
    - cat → [0.9, 0.1, 0.3, −0.2]
    - dog → [0.85, 0.15, 0.25, −0.1]
    - apple → [−0.6, 0.8, 0.05, 0.2] Cosine similarity(cat, dog) will be high, cosine similarity(cat, apple) will be low, illustrating semantic clustering.

Positional encoding details

- **Sinusoidal encoding**: fixed vectors using sinusoids of different wavelengths so models can extrapolate to longer sequences.
- **Learned positional embeddings**: add learned vectors per position; flexible but limited to trained length unless extrapolated.
- **Relative positional encodings**: encode relative distances between tokens rather than absolute positions, helpful for generalization to longer sequences.

Practical tips for embeddings

- Normalize or scale embeddings when needed to stabilize training.
- Use subword tokenization for languages with large vocabularies or morphological complexity.
- Consider using pretrained embeddings or models as initialization for downstream tasks to reduce required data and compute.

Self-Attention Mechanism and Math

Conceptual summary

- Self-attention computes a new representation for each token by comparing it with every other token via learned linear projections called Query, Key, and Value. Attention scores determine how much information each other token contributes to the target token’s updated vector.

Mathematical formulation

- Given token embeddings H = [h1, h2, …, hT] with hi ∈ ℝ^d, define linear projections:
    - Q = H W_Q, K = H W_K, V = H W_V with W_Q, W_K, W_V ∈ ℝ^{d×d_k}.
- Scaled dot-product attention:
    - scores = Q K^T / √d_k

    - weights = softmax(scores)
    - Attention output = weights V.
- Multi-head attention: split projections into H heads with smaller d_k per head, compute attention per head, concatenate outputs, then project back to original dimension.

Step-by-step numeric example

- Suppose d_k = 2, tokens A, B have projected vectors:
    - Q_A = [1, 0.5], K_B = [0.8, 0.2], V_B = [0.6, 0.1]
    - score_AB = Q_A · K_B = 1*0.8* + *0.5*0.2 = 0.9
    - If scores with other tokens are [0.9, 0.4, −0.1], softmax → weights ≈ [0.60, 0.30, 0.10].
    - Output for A = 0.60*V_token1* + *0.30*V_token2 + 0.10*V_token3, so A's new vector is dominated by token1's value.

Masked attention and generation

- For autoregressive generation, apply a causal mask to scores so each position only attends to previous positions and itself, preserving the left-to-right generation property.

Multi-head intuition

- Different heads specialize: one head may track syntactic dependency, another coreference links, another local phrasing patterns. Combining heads increases representational power.

Complexity and scaling

- Self-attention cost per layer is O(T^2 · d) for sequence length T. For long sequences this motivates sparse attention, local windows, hierarchical attention, or efficient approximations to reduce memory and compute.

Visualization and interpretability

- Attention matrices (weights) can be visualized as heatmaps showing which tokens focus on which others. These visualizations often reveal meaningful linguistic patterns such as subject-verb dependencies, entity linking, or anaphora resolution.

Training dynamics and stability

- Layer normalization and residual connections are critical to stabilize deep stacks of attention layers. Learning-rate warmup followed by decay and Adam-style optimizers are common best practices.

Applications Examples Diagram Guidance and Document Structure

Practical applications and worked examples

- **Machine translation**
    - Workflow: tokenize input sentence → embeddings + position → encoder self-attention stacks → decoder with masked self-attention + cross-attention to encoder outputs → generate translation token-by-token.

    - Example: English “She left the meeting early” → cross-attention helps decoder choose correct verb tense and pronoun in the target language.
- **Summarization**
    - Encoder condenses long document into contextual vectors; decoder attends to salient positions to produce a short summary. Attention maps often highlight sentence-level salience for key facts.
- **Question answering**
    - Model attends to passage spans relevant to the question; attention weights identify answer-bearing sentences and tokens.
- **Code generation and completion**
    - Self-attention captures long-range dependencies in code such as variable definitions and later uses; positional encodings help reason about block structure.
- **Vision and multimodal tasks**
    - Vision Transformers treat image patches as tokens; attention captures relationships across spatial locations and combines with text embeddings in multimodal models.